ADVERTIMENT. La consulta d'aquesta tesi queda condicionada a l'acceptació de les següents condicions d'ús: La difusió d'aquesta tesi per mitjà del servei TDX (www.tesisenxarxa.net) ha estat autoritzada pels titulars dels drets de propietat intel·lectual únicament per a usos privats emmarcats en activitats d'investigació i docència. No s'autoritza la seva reproducció amb finalitats de lucre ni la seva difusió i posada a disposició des d'un lloc aliè al servei TDX. No s'autoritza la presentació del seu contingut en una finestra o marc aliè a TDX (framing). Aquesta reserva de drets afecta tant al resum de presentació de la tesi com als seus continguts. En la utilització o cita de parts de la tesi és obligat indicar el nom de la persona autora.

ADVERTENCIA. La consulta de esta tesis queda condicionada a la aceptación de las siguientes condiciones de uso: La difusión de esta tesis por medio del servicio TDR (www.tesisenred.net) ha sido autorizada por los titulares de los derechos de propiedad intelectual únicamente para usos privados enmarcados en actividades de investigación y docencia. No se autoriza su reproducción con finalidades de lucro ni su difusión y puesta a disposición desde un sitio ajeno al servicio TDR. No se autoriza la presentación de su contenido en una ventana o marco ajeno a TDR (framing). Esta reserva de derechos afecta tanto al resumen de presentación de la tesis como a sus contenidos. En la utilización o cita de partes de la tesis es obligado indicar el nombre de la persona autora.

WARNING. On having consulted this thesis you're accepting the following use conditions: Spreading this thesis by the TDX (www.tesisenxarxa.net) service has been authorized by the titular of the intellectual property rights only for private uses placed in investigation and teaching activities. Reproduction with lucrative aims is not authorized neither its spreading and availability from a site foreign to the TDX service. Introducing its content in a window or frame foreign to the TDX service is not authorized (framing). This rights affect to the presentation summary of the thesis as well as to its contents. In the using or citation of parts of the thesis it's obliged to indicate the name of the author

Department of Signal Theory and Communications
Universitat Politècnica de Catalunya

Computing Resource Management in Software-Defined and Cognitive Radios

DISSERTATION

Vuk Marojevic

Dr. Antoni Gelonch
(Ph.D. Supervisor)

Barcelona, July 2009

I would like to thank the Radio Communications Research Group of the UPC for supporting my Ph.D. studies, which led to the elaboration of this thesis. I also thank the Mobile and Portable Radio Research Group of Virginia Tech for their support during my short term visiting scholar in 2007.

This work has been supported by the CICYT (Spanish National Science Council) under grants TIC2003-08609 and TEC2006-09109, which are partially financed from the European Community through the FEDER program, and the AGAUR agency of the Catalonian Government.

Summary

Our research aims at contributing to the evolution of modern wireless communications and to the development of software-defined radio (SDR) and cognitive radio, in particular. It promotes a general resource management framework that facilitates the integration of computing and radio resource management. This dissertation discusses the need for computing resource management in software-defined and cognitive radios and introduces an SDR computing resource management framework with cognitive capabilities. The hard real-time computing requirements of software-defined digital signal processing chains (SDR applications), the associated radio propagation and quality of service (QoS) implications, and heterogeneous multiprocessor platforms with limited computing resources (SDR platforms) define the context of these studies.

We examine heterogeneous computing techniques, multiprocessor mapping and scheduling in particular, and elaborate a flexible framework for the dynamic allocation and reallocation of computing resources for wireless communications. The framework should facilitate partial reconfigurations of SDR platforms, dynamic switches between radio access technologies (RATs), and service and QoS level adjustments as a function of the environmental conditions. It, therefore, assumes the facilities of the *platform and hardware abstraction layer operating environment* (P-HAL-OE).

We suggest a modular framework, distinguishing between the computing system modeling and the computing resource management. Our modeling proposal is based on two computing resource management techniques, which facilitate managing the strict timing constraints of real-time systems. It is scalable and can account for many different hardware architectures and computing resource types. This work focuses on processing and interprocessor bandwidth resources and processing and data flow requirements.

Our computing resource management approach consists of a general-purpose mapping algorithm and a cost function. The independence between the algorithm and the cost function facilitates implementing many different computing resource management policies. We introduce a dynamic programming based algorithm, the t_w-mapping, where w controls the decision window. We present a general and parametric cost function, which guides the mapping process under the given resource constraints. An instance of it facilitates finding a mapping that meets all processing and data flow requirements of SDR applications with the available processing and bandwidth resources of SDR platforms. Several SDR reconfiguration scenarios and analyses based on simulations demonstrate the suitability and potentials of our framework for a flexible computing resource management.

We extend our SDR computing resource management concepts to the cognitive radio context. The two primary objectives of cognitive radio are *highly reliable communications whenever and wherever needed* and *the efficient use of the radio spectrum*. We formulate a third objective as *the efficient use of computing resources*. We analyze the cognitive capabilities of our framework─the cognitive radio's interface to SDR platforms─and indicate the potentials of our *cognitive computing resource management* proposal.

The cognitive computing resource management needs to be coordinated with the radio resource management. We, therefore, introduce the *joint resource management* concept for cognitive radios. We

present three cognitive cycles and discuss several interrelations between the radio, computing, and application resources, where application resources refer to the available SDR and user applications. Our approach potentiates flexibility and facilitates radio against computing resource tradeoffs. It promotes cognition at all layers of the wireless system for a cooperative or integrated resource management that may increase the performance and efficiency of wireless communications.

Resumen

El objetivo de las investigaciones que se están llevando a cabo dentro del grupo de investigación es contribuir a la evolución de las radiocomunicaciones modernas y, en particular, al desarrollo de los conceptos software radio (SDR) y cognitive radio. El planteamiento general es el de extender la flexibilidad global del sistema de comunicaciones planteando la definición y desarrollo de un entorno en el que pudiesen explorarse las relaciones entre la computación y las prestaciones del sistema de comunicaciones móviles facilitando la integración de los recursos de computación con los recursos radio.

Dentro de este marco, la presente tesis plantea la discusión de la necesidad de la gestión de los recursos de computación en entornos SDR y cognitive radio y define un entorno de operación que asume las características especificas del concepto SDR a la vez que incorpora capacidades cognitivas en la gestión de los recursos de computación de las plataformas que den soporte a las nuevas generaciones de sistemas móviles. Los estrictos requerimientos de procesado en tiempo real de las cadenas de procesado digital de la señal definidas por software (aplicaciones SDR), las implicaciones asociadas con la propagación radio y el concepto de calidad de servicio (QoS) y plataformas heterogéneas de múltiples procesadores con recursos de computo limitados (plataformas SDR) definen el contexto de estos estudios.

Se examinan técnicas de cómputo de propósito general para definir un entorno de operación que fuese capaz de asignar de forma flexible y dinámica los recursos de cómputo necesarios para facilitar las radiocomunicaciones a los niveles de QoS deseados. Ello debería facilitar los cambios dinámicos de una tecnología de acceso radio a otra, permitiendo el ajuste del tipo de servicio o calidad de servicio en función de las preferencias de los usuarios y las condiciones del entorno. Dicho entorno de operación asume las potencialidades del *platform and hardware abstraction layer operating environment* (P-HAL-OE).

La estructura del entorno de operación se define de forma modular y consiste en un modelado genérico y flexible de las plataformas de computación SDR y en una gestión de recursos de computación abierta y capaz de ajustarse a diferentes objetivos y políticas. En el trabajo se exponen dos técnicas de gestión que pretenden asegurar la consecución estricta de los límites temporales típicos de los sistemas en tiempo real. En cuanto al modelado, este es escalable y capaz de capturar un amplio abanico de arquitecturas hardware y recursos de computación. En el presente trabajo nos centramos en los recursos y requerimientos del procesado y transferencia de datos.

Se introduce un algoritmo de mapeo genérico e independiente de la función de coste. La independencia entre el algoritmo y la función de coste facilita la implementación de diferentes políticas de gestión de recursos computacionales. El t_w-mapping es un algoritmo basado en *dynamic programming*, donde w controla la ventana de decisión. Se presenta una función de coste genérica y parametrizable que permite guiar el proceso de gestión de los recursos. Una instancia de ella facilita encontrar una solución al proceso de asignación de recursos que cumpla todos los requerimientos de procesado y trasferencia de datos de las aplicaciones SDR con los recursos disponibles de las plataformas SDR. Diferentes escenarios y varios análisis basados en simulaciones demuestran la adecuación del entorno de trabajo definido y desarrollado, así como sus potencialidades para una gestión flexible de los recursos de cómputo.

Se extienden los conceptos mencionados previamente para entornos cognitive radio. Los principales objetivos del concepto cognitive radio son la disponibilidad de comunicaciones altamente robustas en cualquier lugar y momento en que sean necesarias y el uso eficiente del espectro. Como tercer objetivo formulamos *el uso eficiente de los recursos de cómputo*. Analizamos las capacidades cognitivas de nuestro entorno de operación─la interfaz del sistema cognitive radio a las plataformas SDR─y resaltamos las potencialidades de nuestra propuesta de *gestión cognitiva de los recursos computacionales*.

Dicha gestión cognitiva de los recursos computacionales plantea una integración con la gestión de los recursos radio. Para ello introducimos el concepto de *gestión de recursos conjunta* para entornos cognitive radio. Se presentan tres ciclos cognitivos y se discuten algunas interrelaciones entre los recursos radio, de cómputo y de aplicación, donde los recursos de aplicación se refieren a las aplicaciones SDR y de usuario disponibles. Nuestra propuesta de gestión de recursos conjunta potencia la flexibilidad y facilita los intercambios entre recursos radio y de computación.

Contents

Abbreviations

2G	Second generation wireless systems
3G	Third generation wireless systems
3GPP	3rd Generation Partnership Project
4G	Fourth generation wireless systems
ADC	Analog-to-digital converter
AE	Application environment
ALOE	Abstraction layer and operating environment
API	Application programming interface
ASIC	Application-specific integrated circuit
B3G	Beyond 3G
BER	Bit error rate
BSLA	Baseline algorithm
CCC	Communication-to-computation correlation
CCR	Communication-to-computation ratio
CE	Computing environment
CF	Core framework
COMM	Communication
COMP	Computation
CORBA	Common object request broker architecture
COTS	Common of-the-shelf
CP	Control processor
CPU	Central processing unit
CRC	Cyclic redundancy check
CRM	Computing resource management
CRRM	Common radio resource management
CU	Computing unit
DAC	Digital-to-analog converter
DAG	Directed acyclic graph
DCA	Dynamic channel allocation
DCP	Dynamic critical-path
DLS	Dynamic level scheduling
DPS	Dynamic priority scheduling
DSA	Dynamic spectrum allocation
DSP	Digital signal processor
ET	Execution time
FD	Full-duplex
FFT	Fast Fourier Transform
FIFO	First-in-first-out
FISC	Flexible integrated system capability

FlexNets	Flexible wireless systems and networks
FPGA	Field-programmable gate array
GB	Gigabyte
GPP	General-purpose processor
GPRS	General packet radio system
GSM	Global system for mobile communications
HD	Half-duplex
HIPERLAN	High performance radio LAN
HLFET	Highest level first with estimated times
HSDPA	High speed downlink packet access
IDL	Interface definition language
IEEE	Institute of electrical and electronics engineers
IF	Intermediate frequency
IRM	Integrated resource management
IRMA	Integrated resource management algorithm
ITU	International Telecommunication Union
JARM	Joint application resource management
JCRM	Joint computing resource management
JRARM	Joint radio application resource management
JRRM	Joint radio resource management
JUARM	Joint user application resource management
JTRS	Joint Tactical Radio System
kbps	Kilo bits per second
KUAR	Kansas University Agile Radio
LAN	Local area network
LGDF	Large grain data flow
MAIL	Maximum allowable increase in load
MAC	Multiply accumulate (operation)
MB	Megabyte
Mbps	Mega bits per second
MBPTS	Megabits per time slot
MCP	Modified critical path
MOPS	Million operations per second
MOPTS	Million operations per time slot
MPRG	Mobile and Portable Radio Research Group
MP-SoC	Multiprocessor system on a chip
MT	Mobile terminal
mWPS	Milliwatt per second
NoC	Network on a chip
Node B	3GPP term for a base station
NP	Nondeterministic polynomial time
OE	Operating environment
OFDM	Orthogonal frequency division multiplexing
OMG	Object management group
ORB	Object request broker
OS	Operating system
OSI	Open system interconnection
OSSIE	Open source SCA implementation embedded
PE	Processing element
PHAL	Platform and hardware abstraction layer
P-HAL-OE	PHAL operating environment
POSIX	Portable operating system interface
P-SCH	Primary synchronization channel
QoS	Quality of service
Q-SCA	QoS-enabled SCA
RAE	Radio application environment
RAN	Radio access network
RARM	Radio application resource management
RAT	Radio access technology
RE	Radio environment

RF	Radio frequency
RKRL	Radio Knowledge Representation Language
RRM	Radio resource management
RT-CORBA	Real-time CORBA
RTOS	Real-time operating system
SAD	Software assembly descriptor
SCA	Software communications architecture
SCD	Software component descriptor
SE	Service environment
SDF	Synchronous data flow
SDR	Software-defined radio
SDR-BS	SDR base station
SDR-MT	SDR mobile terminal
SNR	Signal-to-noise ratio
SoC	System on a chip
SPD	Software package descriptor
SPTS	Seconds per time slot
TANH	TDS algorithm for network of heterogeneous systems
TDS	Task duplication based scheduling
TS	Time slot
UAE	User application environment
UARM	User application resource management
UE	User equipment
UMTS	Universal mobile telecommunications system
USB	Universal serial bus
USRP	Universal Software Radio Peripheral
UTRAN	UMTS terrestrial radio access network
VM	Virtual machine
WiMAX	Worldwide interoperability for microwave access
WCDMA	Wideband code division multiple access
WLAN	Wireless local area network
WPAN	Wireless personal area network
XML	Extensible markup language

1

Introduction

1.1 Modern Radio Communications

Except for the word *radio* in *radio communications*, today's wireless communications have little in common with the radio introduced by Marconi more than a century ago. Wireless communications are nowadays part of our professional and private life. Their manifold utility ranges from providing basic communications services to interactive multimedia entertainment. Universities and companies, including telecommunications specialist as well as computer scientists, steadily strive for evolving wireless communications. We approach a wireless world that, in many senses, increases our quality of life.

The coexistence of radio access technologies (RATs) characterizes modern wireless communications. The emerging 3G systems, for example, neither pretend to substitute the worldwide available access to 2G services nor directly compete with wireless local or personal area networks (WLANs or WPANs). These RATs rather complement each other: 2G systems, such as the global system for mobile communications (GSM) and the general packet radio system (GPRS), offer worldwide roaming for carrying voice traffic and low-volume data. 3G systems─the universal mobile telecommunication system (UMTS) and its American and Asian predecessors or derivatives─offer higher data rates for many classes of (multimedia) services. WLANs locally offer very high data rates at lower cost for accessing global information, whereas WPANs wirelessly interconnect personal devices. This view of cooperative rather than competitive radio access networks (RANs) facilitates providing personalized services and a ubiquitous wireless access [1].

Beyond 3G (B3G) is the common term that describes today's coexistence of RATs. It characterizes a heterogeneous radio environment where the user, and not the radio, takes the central stage. That is, the traditionally RAT-driven wireless communications become service-driven and personalized wireless communications with quality of service (QoS) differentiations. As RATs evolve over time, today's state-of-the-art RATs may become obsolete some day. The trend, however, indicates that only few RATs will actually be replaced so that the radio environment will remain or become even more heterogeneous.

The latest radio standards increase the spectral efficiency and facilitate a more flexible use of radio resources. UMTS, for example, allows for more flexible channel allocations than 2G systems. This makes variable data rates and bandwidth-on-demand possible. Streams with different bit rates can, moreover, be multiplexed and transmitted together. Flexible bit rates efficiently support multimedia applications, possibly consisting of voice, video, and file transfer components [11]. The high speed downlink packet access (HSDPA) and the worldwide interoperability for microwave access (WiMAX), the successors of UMTS and WLAN, add additional features for increasing the spectral efficiency and service diversity.

As new radio standards emerge, the radio operators desire reusing existing radio infrastructure. Similarly, the introduction of a new user service cannot imply the need for a new mobile terminal. Designing multipurpose infrastructure and mobile terminals for integrating existing and future radio standards is challenging. Multimode hardware designs may be economically feasible today, but will soon become obsolete because of their limited flexibility. A software solution should then be considered, where software

at least partly defines the radio transmission and reception modes of radio equipment. This is known as software-defined radio (SDR). Apart from the flexibility of using one or another air interface and providing one or another service, SDR facilitates infrastructure or hardware upgrades and signal processing or software updates.

SDR introduces flexibility to wireless communications, whereas cognitive radio adds intelligence. Cognitive radio facilitates a situation dependent allocation and use of resources for wireless communications. It can automate the reconfiguration process of SDR equipment for optimally using the available resources at a given place and time. SDR and cognitive radio together offer the required flexibility and intelligence for the personalized and omnipresent service provisioning. The rest of this chapter briefly reviews both concepts and indicates the current SDR and cognitive radio research, focusing on resource management.

1.2 Software-Defined Radio

In the early 90s Mitola envisaged radio transmitters and receivers (transceivers) that implement the entire signal processing chain in software (Fig. 1.1a). He coined this vision *software radio* [3], [4]. Software radio describes multistandard, multiservice, and multiband radio systems, which are software-reconfigurable or reprogrammable. It promises to become a pragmatic solution to the variety of available and incompatible radio standards [6]. The technological difficulty for digitalizing radio frequency (RF) signals [7], [8], [9] led to the introduction of SDR [10]. SDR may be considered as a generalization of software radio. It characterizes a transceiver that implements one or more signal processing blocks in software. Since digitalization usually takes place at the intermediate frequency (IF) stage (Fig. 1.1b), part of the IF, baseband, and higher layer processing can then be implemented in software [11].

SDR introduces flexibility to wireless systems: It permits adjusting or switching a terminal's RAT implementation for adapting to changes in the heterogeneous radio environment. As opposed to traditional radio communications, the software that runs on an SDR platform defines its radio functionality. SDR platform stands for a software-programmable computing equipment, including handset transceivers, base stations, and core networks. SDR application or waveform refers to (part of) a RAT-specific digital signal processing chain. Reconfiguring an SDR platform to execute another SDR application could then change the deployed radio standard for establishing a radio communications link. Dynamic RAT switches or modifications during communications sessions are also envisaged.

For about a decade, SDR-related research along the whole line between the mobile terminal transceiver and the core network has been ongoing [12], [13], [14], [15]. It is motivated by the evolution of information technology: The introduction of new RATs, such as UMTS or the IEEE 802.11 family of WLANs, the required compatibility with the existing RATs, including GSM and GPRS, and the increasing demand for new and differentiated user services call for flexible transceiver solutions. Therefore, the flexibility of general purpose processors (GPPs), digital signal processors (DSPs), field-programmable gate arrays (FPGAs), picoArrays [25], networks on a chip (NoCs) [26], or multiprocessor systems on a chip (MP-SoCs) [27] is becoming more important than the energy efficiency of application-specific integrated circuits (ASICs) [28], [29], [30]. Today's reconfigurable devices offer high computing capacities at moderate

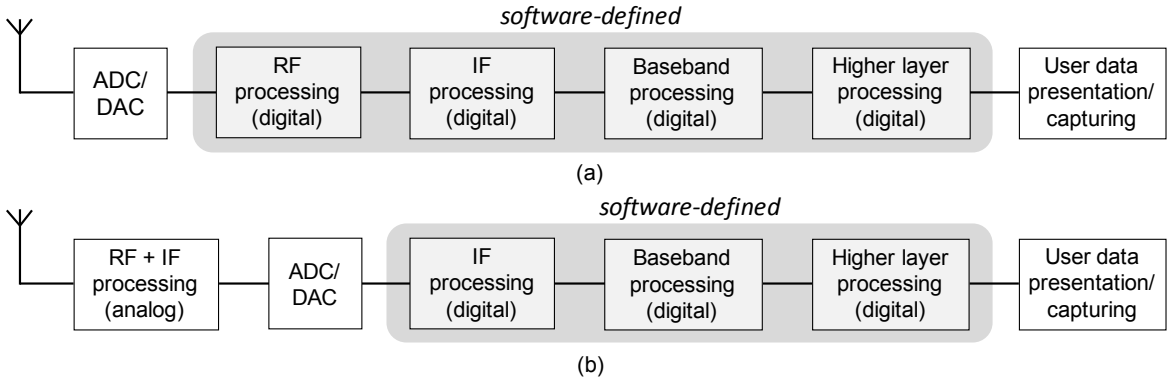


Fig. 1.1. Conceptual software radio (a) and software-defined radio (b) transceivers.

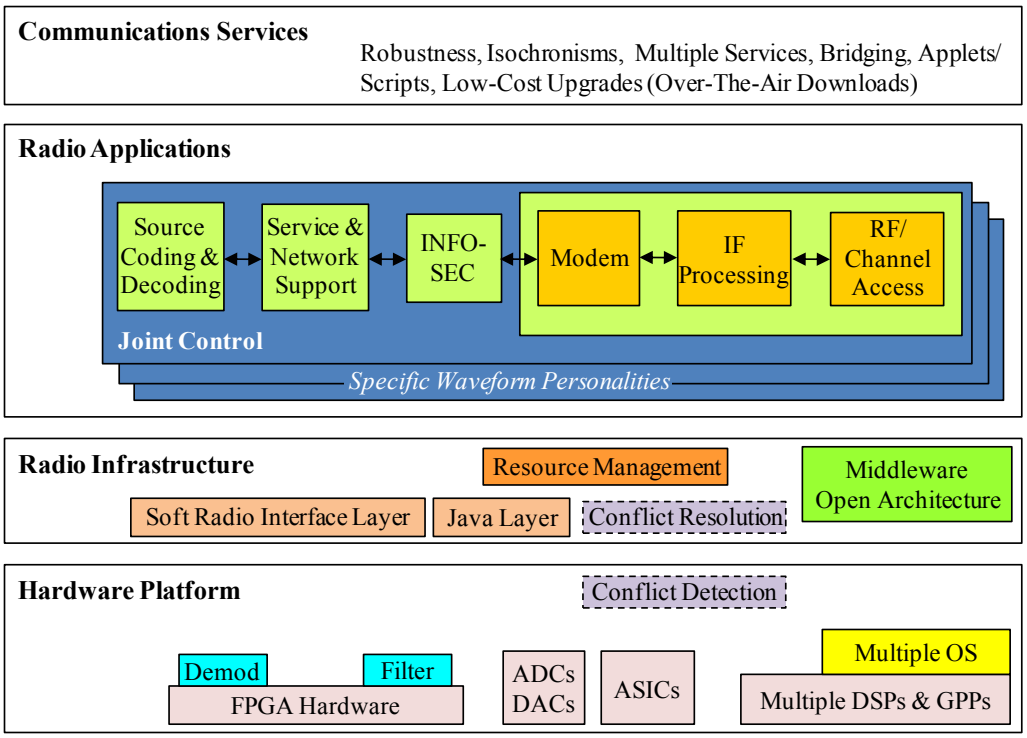


Fig. 1.2. Layered software radio architecture [5].

power consumptions. As a result, we observe a tendency of expanding the digital and reconfigurable radio part while reducing the analog and nonreconfigurable circuits.

Fig. 1.2 shows Mitola's *layered software radio architecture* [5]. It does not specify a particular SDR implementation, but, rather, a general framework for SDR communications. Its four layers are independent to some extent. The *communications services* range from simple phone calls to interactive multimedia services. They are steadily redefined or extended as a function of user demands and market success. *Radio applications* stand for SDR applications or waveforms. They define the signal processing and over-the-air transmission modes for adequate service and QoS provisioning. The *radio infrastructure* is the abstract execution environment for radio applications. It may feature middlewares, virtual machines (VMs), resource managers, and conflict solvers. The *hardware platform* typically consists of a set of heterogeneous processors, data converters, and operating systems (OSs).

The figure indicates that radio applications and hardware platforms are heterogeneous. That is, a radio application consists of several modules that provide different functionalities. These modules assemble the waveform, but each one of them may be more or less complex and, thus, require more or less computing resources. Hardware platforms correspondingly consist of different types of processing elements (PEs), ranging from ASICs to GPPs. To flexibly support dynamic software and hardware environments, the resource management module needs to be independent from a particular radio application and from a certain hardware platform. It is therefore located within the platform and application-independent radio infrastructure (Fig. 1.2).

1.3 Cognitive Radio

In the late 90s and less than a decade after introducing software radio, Mitola coined *cognitive radio* [114]. His dissertation [115] initiated worldwide cognitive radio research [116], [117], [118], [119] and led to many different interpretations of the cognitive radio concept. A cognitive radio system is commonly considered as an intelligent wireless communication system aiming at the efficient usage of radio resources [116], [118]. It is, however, a much broader and more powerful concept [115], [133]. Mitola defines it as "a goal-driven framework in which the radio autonomously observes the radio environment, infers contexts, assesses alternatives, generates plans, supervises multimedia services, and learns from its mistakes" [137]. We suggest the following, similar definition:

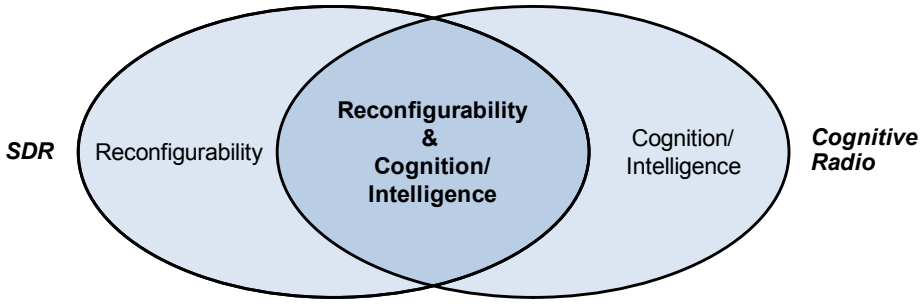


Fig. 1.3. Software-defined versus cognitive radio.

Definition 1 *A* cognitive radio system *is an ambient-aware wireless communications system that can intelligently adjust itself to its operating environment. The system can particularly modify the radio transmission modes or parameters as a function of the resource availabilities, end-to-end QoS requirements, business models, or the like.*

Cognitive radio evolved from SDR [137] although it does not necessarily require a reconfigurable hardware environment (Fig. 1.3). Multimode radio terminals, whose reconfiguration capabilities are limited to mode switches between a limited set of modes, are apt for cognitive radios. The flexibility of software implementations, however, increases the potential of environmentally stimulated radio adjustments: Cognitive radio can, as an intelligent management layer on top of an SDR system, automate and optimize the reconfiguration of SDR equipment. It is *the* application for SDRs.

Fig. 1.4a sketches a simplified interpretation of Mitola's cognition cycle [114], [137]. It consists of the *observe*, *reason and decide*, *learn*, and *act* phases. The environment is observed and analyzed. Decisions are made as a function of these observations and the available knowledge. These decisions are executed and evaluated. This observe-think-act cycle runs continuously. The system gains experience during operation. The general objective is optimizing the use of resources for wireless communications at any time while satisfying the users' service requests and increasing the operators' revenues. Cognitive radio can take advantage of the versatility of the B3G context.

Haykin [116] concretizes Mitola's cognition cycle, considering it for solving radio resource management (RRM) issues. The *basic cognitive cycle* of Fig. 1.4b essentially consists of the *radio-scene analysis* and the *channel-state estimation and predictive modeling* blocks, which process the RF stimuli for the *transmit-power control and spectrum management*. The corresponding actions may affect the radio environment and lead to new RF stimuli [116].

1.4 Resource Management in SDR and Cognitive Radio

Resource management is an important topic of SDR and cognitive radio research. This Section provides a brief overview of some advanced RRM techniques, which are investigated in cognitive radio. We then discuss the need for computing resource management in software-defined and cognitive radios. After examining the related work, we present our resource management outlook as a motivation for this dissertation. We, particularly, indicate some challenges and possibilities of our research and its possible impact on future radio communications as a whole.

1.4.1 Radio Resource Management

Spectrum for radio communications is a limited physical resource. Physical and technological limits restrain the RF carriers to frequencies that range from approximately 0.25 MHz to 6 GHz [137], although the use of higher frequencies is recently explored. The steadily growing bandwidth demand thus creates contention for radio spectrum, a precious radio resource. Spectrum is internationally regulated by the ITU and nationally by governmental agencies [120], [121]. It has traditionally been leased to operators, gaining them exclusive rights to spectrum portions for specific usage. The operators are responsible for efficiently using the acquired radio resources while complying with the specifications imposed by the regulation bodies.

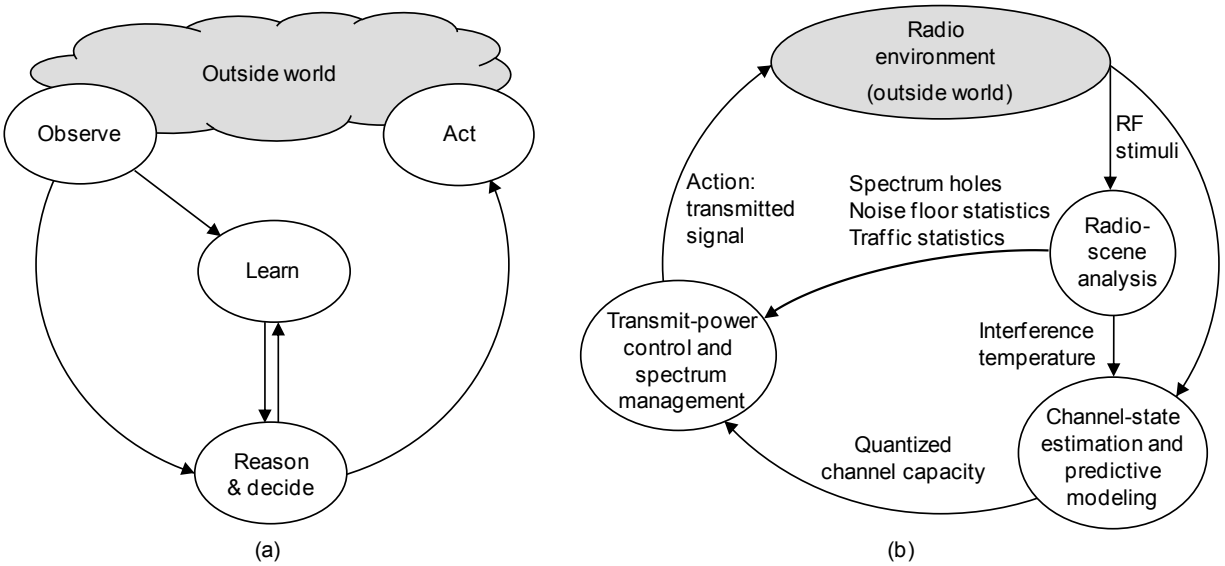


Fig. 1.4. Simplified cognition cycle (a) and Haykin's [116] basic cognitive cycle (b).

RRM is an essential topic in modern wireless communications, which leverage diversity and flexibility and facilitate a higher spectral efficiency. Spectral efficiency is related to system capacity and revenues. This explains the importance of an efficient usage of radio resources.

The basic RRM concept, where each operator manages the radio resources individually for each RAT, has been extended with the introduction of 3G and its coexistence with other radio standards. Common or joint RRM (CRRM or JRRM), for example, refers to the management of radio resources pertaining to different RATs of a single operator [132], whereas dynamic spectrum access implies a more flexible spectrum allocation and management [133]. These techniques are widely investigated in the 3G, B3G, and cognitive radio contexts. We sketch them below.

A) RRM in 3G and B3G

The spectrum auctions for UMTS licenses in Europe have revealed the importance of RRM. (UMTS is the most widely adopted 3G system, which is standardized by the 3GPP [2].) As opposed to 2G systems, the UMTS terrestrial radio access network (UTRAN) can achieve higher spectral efficiencies through advanced coding and multiple access techniques, diversity schemes, and so forth. The wideband code division multiple access (WCDMA) technique, which UTRAN employs, implicitly calls for interference or radio resource management, because many users concurrently transmit over the same frequency channel with possibly different data rates. Manufacturers then have to introduce more sophisticated RRM strategies than those that were used in the past. Service differentiations, the QoS concept, and the possibility of dynamically adapting the resource allocations to the given channel conditions make RRM more powerful but also more challenging [122].

We have already mentioned that the usable spectrum for wireless communications is rigorously divided into frequency bands, which are assigned to different user groups for specific usage. The UMTS spectrum auctions followed this concept. Despite the fixed frequency allocations, a UMTS operator needs to manage various transmission bands and other related radio resources, such as transmission power.

The coexistence of radio standards in B3G leverages the flexibility of choosing a RAT while introducing new RRM challenges. The major challenge consists in introducing advanced RRM strategies or algorithms that operate from a common rather than technology or operator-driven perspective. The sporadic and geographically varying use of spectrum, with some frequency bands being more heavily used than others [142], further argues for inter-frequency and inter-operator RRM.

CRRM or JRRM refers to the management of radio resources of various RATs of a single operator. The CRRM entity therefore coordinates the use of radio resources of different RATs (Fig. 1.5). The 3GPP suggests dividing the radio resources that pertain to an operator into radio resource pools. A radio resource pool contains the available radio resources that pertain to a single RAT within a set of cells. An RRM entity of Fig. 1.5 manages the radio resources of one radio resource pool, whereas the CRRM entity coordinates the management of several resource pools [132].

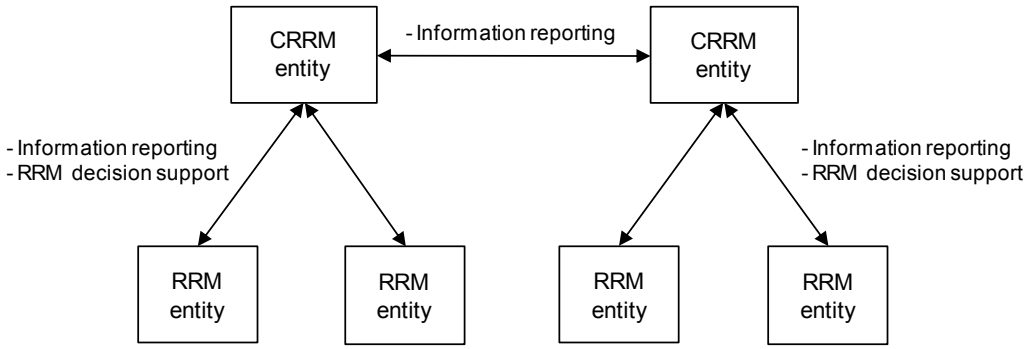


Fig. 1.5. CRRM functional model [132].

B) Dynamic Spectrum Access

Dynamic spectrum access stands for the opposite of the current spectrum regulation and management policies. Zhao and Sadler [133] try to capture its broad significance, providing a taxonomy of dynamic spectrum access, which Fig. 1.6 reproduces. The *dynamic exclusive use model* assumes the fixed spectrum regulation structure that we have today. It though allows to sell and trade spectrum and to freely choose technology (*spectrum property rights* [134]) or to dynamically assign spectrum by exploiting the spatial and temporal traffic statistics of different services (*dynamic spectrum allocation* [135]). The *open sharing* or *spectrum commons model* refers to sharing spectral regions among peer users [136]. The *hierarchical access model* adopts a hierarchical access structure with primary and secondary users. It allows secondary users to access licensed spectrum while not affecting primary user communications. Secondary users can either transmit below the noise floor of primary users (*spectrum underlay*) or occupy temporally unoccupied spectrum (*spectrum overlay*).

Despite these differentiations, the above concepts are closely related. The dynamic spectrum allocation (DSA) and spectrum overlay techniques, for example, try to assign spectrum portions that are momentarily unused (spectrum holes or white spaces) to primary and secondary users, respectively. These methods are appropriate for cognitive radios and are briefly discusses in continuation.

Dynamic Spectrum Allocation

While a static spectrum allocation, which is simple to regulate and manage, was appropriate for wireless communications in the past, it is inefficient for modern wireless communications: The traditionally independent RANs are merging into composite radio networks, basic communication services are replaced by advanced and personalized multimedia services, and a steadily increasing number of users desire an anywhere and anytime wireless connectivity. The time and regional variations of the service and traffic demands reflect another shortcoming of a fixed spectrum allocation. DSA was introduced to overcome these and other inefficiencies and to exploit the potentials of wireless technology [126].

DSA covers a range of research fields, such as *dynamic channel allocation* (DCA), *frequency assignment*, *unlicensed spectrum access*, and *spectrum coexistence*. DCA refers to sharing the radio resources of a single RAN [127], whereas frequency assignments are mostly related to assigning frequencies to base stations as a function of the interference and coexistence constrains [128]. The unlicensed spectrum access approach treats spectrum as an open source that any conforming equipment can use [129]. Spectrum coexistence finally implies different networks using the same frequency bands, such as digital and analog TV.

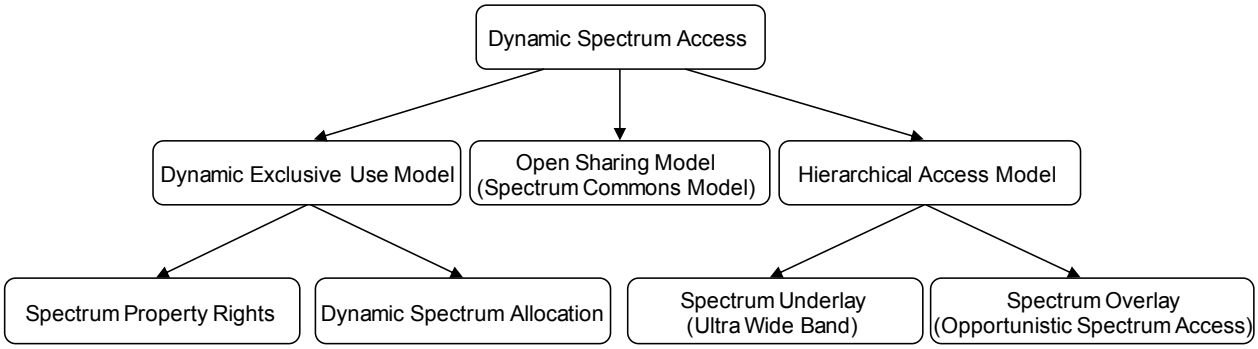


Fig. 1.6. A taxonomy of dynamic spectrum access [133].

Table 1.1 Mobile spectrum pools [137].

Band	RF_{min} (MHz)	RF_{max} (MHz)	W_c	Remarks
Very Low	26.9	399.9	315.21	Long range vehicular traffic
Low	404	960	533.5	Cellular
Mid	1390	2483	930	Personal communication services
High	2483	5900	1068.5	Indoor and RF LANs

DSA is more general than DCA and frequency planning but more organized than the unlicensed access or coexistence methods. It aims at a dynamic spectrum allocation across RANs, facilitating spectrum sharing over space and time. More precisely, the DSA establishes that spectrum is locally and exclusively allocated to services for a limited time. Spectrum can then be allocated as needed, increasing the overall spectral efficiency.

Contiguous and *fragmented DSA* are studied as two practical implementations of DSA, where guard bands separate the spectrum blocks allocated to different RANs [126]. Contiguous DSA implies that, to allocate more bandwidth to one RAN, its spectral neighbors need to release bandwidth. In the fragmented DSA scenario, on the other hand, RANs can occupy any available spectrum hole.

Spectrum Pooling

An opportunistic spectrum access (OSA) includes the opportunity identification, its exploitation and regulation. The opportunity identification process recognizes underused spectrum bands. The opportunity exploitation process then decides if and how the transmission should occur in these bands following some regulation policy, which defines the basic etiquette for secondary users to ensure compatibility with legacy systems. The radio etiquette would, for example, specify that secondary users change bands when primary users enter the bands. The overall objective is exploiting spectrum silent times (in which spectrum portions are underexplored or idle) for secondary usage while protecting primary users and their privilege to access radio resource whenever desired [133].

Spectrum pooling is an OSA technique that was introduced by Mitola [137] and explored by Weiss and Jondral [131]. It refers to short-time spectrum leases that are managed by the cognitive radio itself and priced by the market. Therefore, spectral ranges from different owners are merged into a common pool. Mitola [137] suggested the four spectrum pools of Table 1.1, where W_c is the total bandwidth that could participate in the spectrum pool excluding those bands that are not suitable for pooling, such as satellite, aircraft, and radio navigation bands.

Weiss and Jondral [131] identify OFDM as a candidate modulation for spectrum pooling, because it eases the realization of spectral coexistence between primary and secondary users. In OFDM, a high-rate data stream is divided into multiple low-rate substreams, each modulated on a single subcarrier. Through proper subcarrier spacing, precise synchronization, and rectangular pulse shaping, subcarriers are orthogonal to one another [139]. This facilitates flexible and dynamic allocations of spectrum holes of different bandwidths.

Spectrum pooling as well as other dynamic spectrum access techniques pragmatically approaches a more efficient use of spectrum. The corresponding research results show the advantages of a dynamic spectrum regulation, which is currently evaluated by the regulation authorities. In the long term, this may lead to an open spectrum access [140].

1.4.2 Computing Resource Management

Computing resource management is not a traditional radio communications issue. Traditional radios are designed for accessing a single or a few air interfaces or channels. During the design time of a *hardware-defined radio*, computing resources are assigned to perform specific tasks. During operation, there is little or no need for computing resource management. This is different in SDR, where some computing resources can be reconfigured. The reconfiguration process frees and reassigns computing resources, demapping one software piece and mapping another. This implicitly calls for computing resource management.

SDR facilitates the independent design of software and hardware for wireless communications. Sophisticated real-time services, high data rates, and complex signal processing algorithms entail elevated computing requirements. Software-defined UMTS transceiver implementations, in particular, require high

amounts of memory and processing resources [11], [17]. Multiprocessor execution environments are therefore envisaged for executing even a single-user SDR transceiver [9], [17].

Multiprocessing is a timely research topic and several recent industry success examples, such as the picoArray [31] and the Cell multiprocessor [32], indicate the trend in processing technology for, but not limited to, wireless communications. The modules of an SDR application─essentially the digital signal processing blocks and the data flows between them─then need to be mapped to the distributed and limited computing resources of heterogeneous multiprocessor platforms (Fig. 1.7).

A) Related Work

Several radio standards have been at least partly implemented in software and executed on general-purpose hardware. SDR research has also addressed the partitioning of waveforms for their distributed execution on multiprocessor platforms. Most of these efforts address the mapping of a specific waveform to a particular platform, including the mapping of

- an OFDM baseband processing implementation to a multiprocessor platform [98],
- an OFDM receiver to a MP-SoC [99],
- a WCDMA receiver to a DSP with attached reconfigurable logic [100],
- a HIPERLAN/2 implementation to the MONTIUM processor tile [101],
- a UMTS base station to four PC101 picoArrays using 740 processors [25], and
- the H.264 encoding algorithm to the Cell multiprocessor [102].

Rhiemeier's dissertation [54], on the other hand, studies the processor allocation problem from a rather general perspective. He applies operational research techniques to partition task graphs, which represent waveforms, for their scheduling on a multiprocessor platform. He considers a platform that consists of two identical DSPs connected through a bidirectional bus for studying the implications of bus bandwidth and partitioning algorithm on the execution time of waveforms [53], [54].

Current cognitive radio research focuses on RRM. Computing resource management has not been addressed in this context. Mitola [115], though, talks about self-awareness and introduces a high-level computational model that the system has of itself. The model provides the means for representing the types of processors and their capacities, among others. This information is used for controlling the execution time of the cognition cycle and for detecting software-software and software-hardware incompatibilities [115], although it could be used for computing resource management. Jondral [119] considers the "awareness of processing capabilities for the partitioning or the scheduling of processes" as a technology centric property of cognitive radios. Chapters 2 and 7 examine further related work on computing resource management in SDR and cognitive radio, respectively.

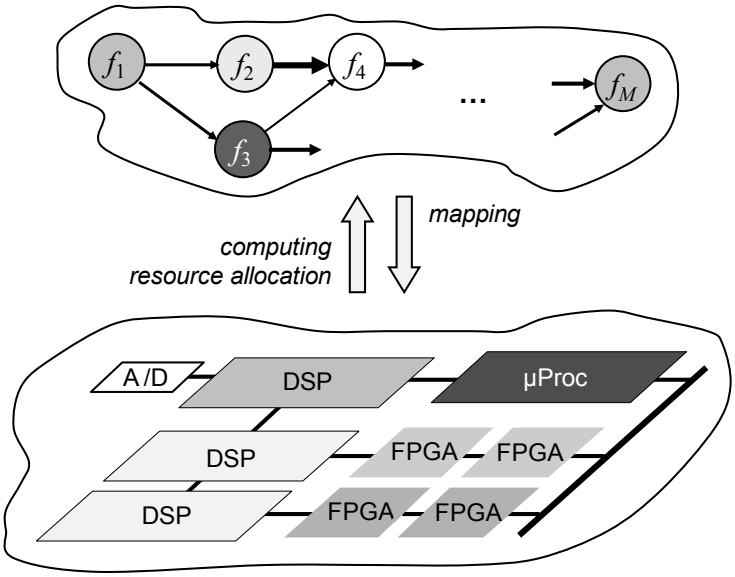


Fig. 1.7. Mapping versus computing resource management.

B) Outlook

The hard real-time computing constraints of SDR applications have implications on the design of SDR platforms and applications as well as on SDR middleware, abstraction layers, and computing resource management solutions. The first generation of SDR mobile terminals (SDR-MTs) will be limited in computing resources, including battery power, and will probably not be capable of supporting more than one RAT implementation at a time. The flexibility of these terminals will then be function of the capabilities of their reconfiguration managers. The computing resources of SDR base stations (SDR-BSs), on the other hand, will be less limited. Nevertheless, an optimization of computing resources would be highly desirable for reducing the operational cost of SDR-BSs. The potentially large number of users and the platforms' high degrees of flexibility, modularity, and reconfigurability make the computing resource management at SDR-BSs equally important though more complex than that at SDR-MTs.

The software challenges include designing efficient implementations of complex signal processing algorithms [8]. Furthermore, software needs to be portable, which has implications on the software design tools. If SDR software needs to be specifically tailored for each hardware platform, software-hardware but also software-software interoperability issues may arise, compromising flexibility. Many SDR solutions are proprietary and hardware or tool-specific software implementations, such as picoChip's SDR transceiver implementations [25], although open source software for programmable processors exist; GNU Radio [97] being the most popular example.

SDR research addresses these (software and hardware) issues and a few practical SDR execution environments have emerged throughout this decade. Theoretical SDR research has gained less attention. We understand an SDR application 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 the 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 think that the introduction of appropriate mapping and scheduling techniques will leverage the design of SDR platforms and applications. *Mapping* (*matching* in heterogeneous computing literature [43]) 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 computing resource management methods.

Wireless or SDR systems, however, reveal specific aspects─essentially regarding flexibility and efficiency─which have not been jointly considered so far:

1. time slot based division of the transmission medium (*radio time slot*),
2. continuous data transmission and reception [34],
3. RAT-specific QoS targets,
4. real-time computing requirements and limited computing resources,
5. different computing constraints and loads for different RATs or radio conditions,
6. dynamic reconfigurations of the protocol stack, either partial or total, and
7. heterogeneous multiprocessor platforms.

These characteristics have implications on the design of the radio infrastructure, the interface between SDR applications and platforms. The system's reconfiguration flexibility and efficiency particularly depends on the computing resource management module. This module is the central part of the radio infrastructure, comparable to the central processing unit (CPU) scheduler within an OS. Its capability to dynamically assign computing resources to computing requirements facilitates the introduction of new (communications) services, the spread of advanced signal processing techniques, the incorporation of hardware extensions, and so forth. We therefore think that computing resource management is essential for the breakthrough of SDRs.

The possibility of dynamically reconfiguring an SDR platform provides full flexibility for adapting an SDR application to the momentary radio environment and QoS demands. SDR can thus be considered as a platform for cognitive radio and advanced RRM [124]. This requires extending current computing resources management approaches toward a framework that links the computing with the radio resources. A cognitive radio system that observes (momentarily) low radio traffic may then give a higher importance to

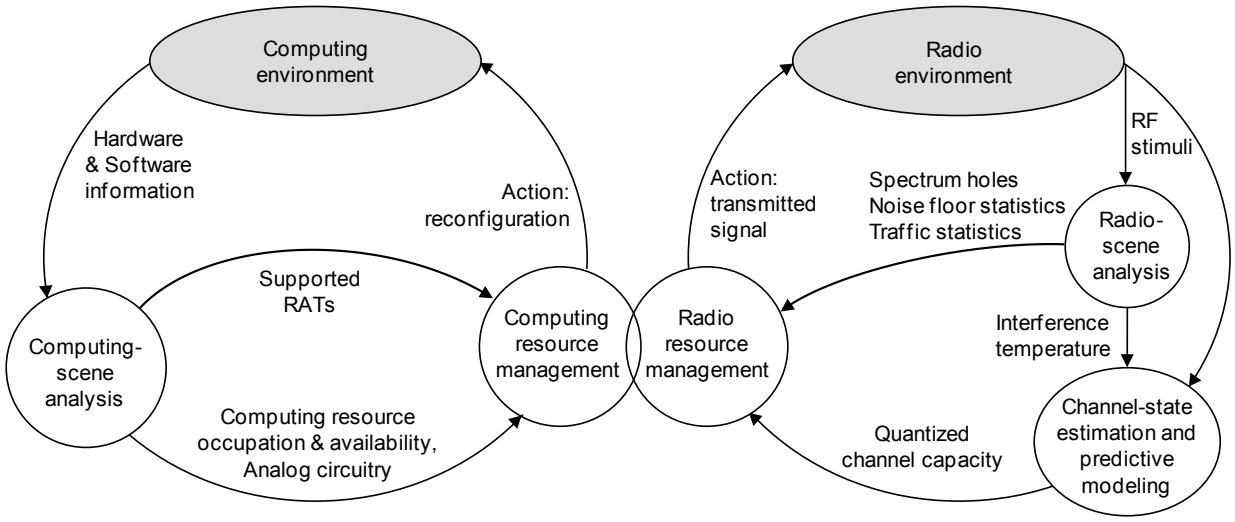


Fig. 1.8. Extended cognitive cycle.

the efficient use of computing resources, reducing the computing complexity and, consequently, power consumption. Similarly, if a non-real-time service would momentarily exhaust a platform's computing capacity, the cognitive radio system could schedule this service at some later time when the limiting conditions change, such as in the event of a battery recharge.

From the above discussion follows that radio resource monitoring and management are not the only issues in cognitive radio. Without the support of computing resources, many advanced RRM techniques cannot be realized. More computing resources are generally required to increase the spectral and transmission power efficiency, in particular. Moreover, future radio communications systems will consider extended models of the entire system and more complex algorithms for reaching higher levels of intelligent management. This evolution calls for more computing resources.

Instead of viewing computing resources and their management as an enabler for cognitive radio, we argue for a cooperative radio and computing resource management. We, therefore, extend the cognitive cycle of Fig. 1.4b, adding another cycle, which monitors and manages the computing resources (Fig. 1.8). This facilitates a synchronized management of the limited radio and computing resources.

Wireless communications is nowadays an integral part of computer engineering (sensor networks and pervasive computing, for example) and computing issues become the more important the further advanced the wireless communications. Our research therefore promotes a higher integration of computing issues in wireless communications and tries to establish the basis for flexible tradeoffs between computing and radio resources.

1.5 Contribution and Outline

Our research aims at contributing to the evolution of modern wireless communications and to the development of SDR and cognitive radio, in particular. It promotes a general resource management framework that facilitates the integration of computing and radio resource management. This dissertation discusses the need for computing resource management in software-defined and cognitive radios and introduces an SDR computing resource management framework with cognitive capabilities. The hard real-time computing requirements of SDR applications, the associated radio propagation and QoS implications, and heterogeneous multiprocessor platforms with limited computing resources define the context of these studies.

We examine heterogeneous computing techniques, multiprocessor mapping and scheduling in particular, and elaborate a flexible framework for the dynamic allocation and reallocation of computing resources for wireless communications. The framework should facilitate partial reconfigurations of SDR platforms, dynamic switches between RATs, and service and QoS level adjustments as a function of environmental conditions.

The context of this dissertation is the *platform and hardware abstraction layer operating environment* (P-HAL-OE), a specific SDR framework. An SDR framework behaves like an OS for radio communications. It has access to physical hardware resources and software repositories and provides an execution

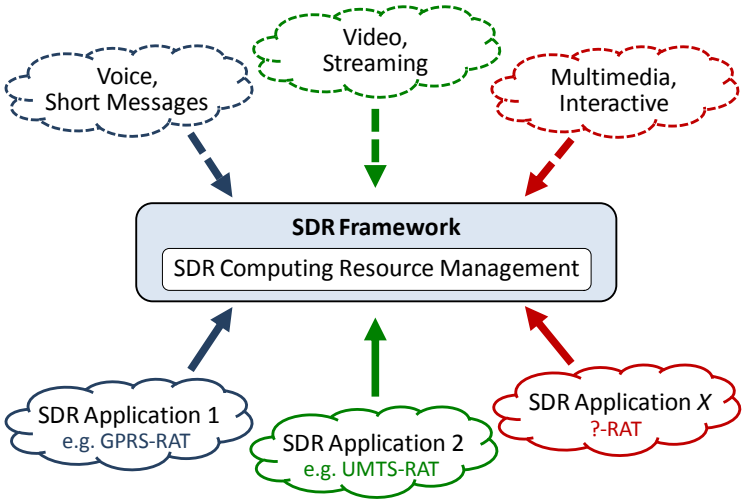


Fig. 1.9. SDR computing resource management context: dynamic mapping of waveforms for a flexible service provisioning.

environment for SDR applications (Fig. 1.9). We review some important SDR frameworks and present the main features of the P-HAL-OE. Chapter 2 also specifies the computing resource management problem and examines the related work.

We suggest a modular framework and distinguish between the computing system modeling and the computing resource management. Fig. 1.10 indicates this modular design approach. The SDR computing system modeling is presented in Chapter 3. It models the computing resources and requirements based on two computing resource management techniques, which facilitate meeting the strict timing constraints of real-time systems. The modeling is scalable and can account for many different hardware architectures and computing resource types. This work focuses on processing and interprocessor bandwidth resources and processing and data flow requirements. Section 3 also introduces several modeling parameters for capturing specific platform and application features.

Chapter 4 presents our computing resource management approach, which consists of a general-purpose mapping algorithm and a cost function. The independence between the algorithm and the cost function facilitates implementing many different computing resource management policies. We introduce a dynamic programming based algorithm, the t_w-mapping, where w controls the decision window. A general and parametric cost function guides the mapping process under the given resource constraints, whereas an instance of it facilitates finding a mapping that meets all processing and data flow requirements of SDR applications with the available processing and bandwidth resources of SDR platforms. This cost function assumes the SDR computing system models of Chapter 3. Chapters 5 and 6 simulate several SDR reconfiguration scenarios and analyze the suitability of our framework and its potentials for a flexible computing resource management.

We extend our SDR computing resource management concepts to the cognitive radio context. The two primary objectives of cognitive radio are *highly reliable communications whenever and wherever needed* and *the efficient use of the radio spectrum* [116]. We formulate a third objective as *the efficient use of*

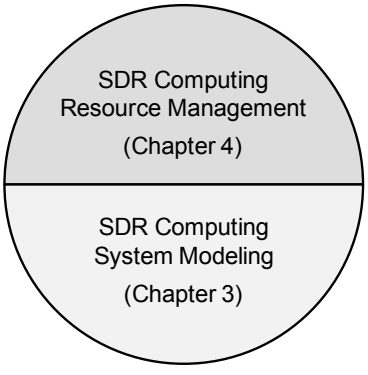


Fig. 1.10. Modular SDR computing resource management framework.

computing resources. The *cognitive computing resource management* then correspondingly extends the currently investigated scope of cognitive radio. Chapter 7 examines the cognitive capabilities of our framework—the cognitive radio's interface to SDR platforms—and indicates the potentials of our proposal.

The cognitive computing resource management needs to be coordinated with the RRM. Chapter 8, therefore, introduces the *joint resource management* concept for cognitive radios. It proposes three cognitive cycles and discusses several interrelations between the radio, computing, and application resources, where application resources refer to the available SDR and user applications. Our approach potentiates flexibility and facilitates radio against computing resource tradeoffs. It promotes cognition at all layers of the wireless system for a cooperative or integrated resource management that may increase the performance and efficiency of wireless communications. Chapter 9 concludes the dissertation with a summary of its main contributions and a brief outlook on future research possibilities.

2

SDR Computing Resource Management Context

2.1 Introduction

This chapter presents the context of our SDR computing resource management framework of Chapters 3 and 4. We first review some important SDR frameworks (Section 2.2). These frameworks abstract the hardware details of SDR platform and constitute the operating environment for SDR applications. We specify the SDR computing environment, formulate the problem, and discuss the scope and assumptions of our solution in Section 2.3. Section 2.4 examines the related work in the general context of heterogeneous computing. We can then concretize the main features that an SDR computing resource management framework should satisfy, while taking into account prior achievements (Section 2.5).

2.2 SDR Frameworks

An execution environment provides a platform for the execution of applications. It basically abstracts hardware resources and provides several functionalities to applications for a controlled use of these resources. Execution environments for general purpose computing platforms and applications include common OSs, such as UNIX or POSIX (portable operating system interface) [20], higher level abstractions or virtual machines (VMs) [20], object oriented architectures, such as the common object request broker architecture (CORBA) [21], and grid computing environments [23]. While conceptually appropriate for the SDR computing context, these environments do not specifically address digital signal processing issues and introduce timing, memory, and power overheads that make it difficult to meet the hard real-time computing constraints of SDRs. Therefore, general-purpose computing knowledge and experience has been adapted to the digital signal processing world and SDR frameworks emerged.

Several SDR frameworks exist. The Joint Tactical Radio System's (JTRS's) software communications architecture (SCA) [87], for example, is an SDR framework specification that is based on CORBA. Since SCA is the most widespread SDR research project, we may classify SDR frameworks as SCA and non-SCA based. This Section briefly reviews the SCA, the QoS-enabled SCA (Q-SCA), and the P-HAL-OE. We discuss the frameworks' principal characteristics and evaluate their computing resource management capabilities. Section 2.2.4 finally summarizes a few additional SDR frameworks; many more exist.

2.2.1 SCA

The JTRS is the US military software radio research program and the SCA is its baseline architecture [87]. The SDR Forum, industry, and academia are promoting, developing, and implementing the SCA. The

SDR Forum is an independent and nonprofit organization [15]. It is attempting to fill the role of a software radio standards body with the mission of fostering software radios in a flexible wireless market. Its members are companies, universities, and other associations with experience or interest in SDRs. The SDR Forum proposed its own SDR framework, which is based on the JTRS-SCA.

The Mobile and Portable Radio Research Group (MPRG) of Virginia Tech is developing the open source SDR framework *OSSIE* (*open source SCA implementation embedded*) [89]. OSSIE is primarily intended to enable research and education in SDR and wireless communications. The software package, which is downloadable from [89], includes an SDR core framework based on the JTRS-SCA, tools for rapid development of SDR components and waveforms, and an evolving library of pre-built applications.

Here we briefly present the SCA and its computing resource management capabilities, rather than examining the different SCA implementations. If not stated otherwise, we discuss the JTRS-SCA release 2.2 [88].

A) Objectives

The SCA was introduced to reduce software manufacturing costs by reducing the development time through the reusability and portability of waveforms. The initial goals of the SCA were [14]:

1. function as a multiband multimode radio,
2. be interoperable with all JTRS domains (airborne, fixed/ maritime, vehicular, dismounted, and hand-held),
3. be compatible with legacy systems,
4. support the insertion of new technologies,
5. support advanced networking features, and
6. use primarily commercial off-the-shelf (COTS) components.

These objectives were generalized by the SDR Forum's approach, specifying

1. the domain independence across all commercial, civil, and military implementations, and
2. the support for commercial, civil, and military waveforms.

The SCA thus approaches a common architecture with clear interface definitions for easing the design of flexible and interoperable SDR execution environments and portable waveforms. It encourages the extensive use of object-oriented methodologies. It emphasizes on continuous validations, where the specifications are validated and refined during the research, prototyping, and implementation processes.

B) Architecture

The basic architecture comprises a software and a hardware framework as well as rules that govern the software and hardware implementations. The software framework defines the software structure and the interactions between software components. The hardware framework describes the hardware classes and subclasses and not individual and domain-specific hardware choices. The rules relate to specific implementations. They specify interfaces, form factors, software development languages, and so forth. They particularly define specific software and hardware requirements that must be met for an implementation to be SCA compliant.

The general software stack of the SCA consists of the *core framework* (CF), the CORBA object request broker (ORB), and the POSIX-based real-time operating system (RTOS) (Fig. 2.1). The CF describes the interfaces. It provides an abstraction of the software and hardware layers for waveform developers. The CORBA ORB facilitates the communication between software objects. Therefore, each object must provide an interface written in the object management group (OMG) interface definition language (IDL). This encapsulates the actual functionality of a CORBA object behind its interface.

All software and hardware layers of the SCA are independent and can be independently modified. Interfaces between these layers are created from specified and publicly maintained standards, applying object-oriented methodologies: A waveform, for instance, is described as an object, consisting of several functional objects (modules or components). Objects contain private data and public member functions. These functions specify the methods for interactions between objects. Software objects can describe and

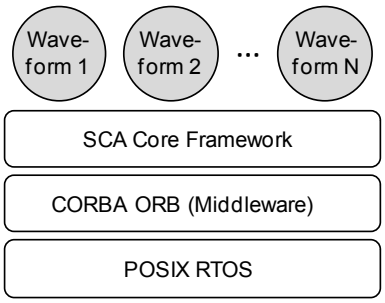


Fig. 2.1. The SCA software stack.

control hardware devices. Fig. 2.2 illustrates a more detailed diagram of the SCA software framework. It shows a generic framework with only a few objects being hardware-specific.

Resource stands for a software object that features a set of radio functionalities. It is the fundamental software element of the SCA. All non-CF software structures are Resources. A Resource may provide direct control over a hardware device or deliver some software service. Resources can communicate with one another using *ports*—abstract accesses to communication channels—and establish complex and dynamic interdependencies. *Adapters* are special Resources that are used to access a non-CORBA-compliant interface of a nonstandard Resource. Adapters facilitate using a wide variety of devices as well as legacy code.

The CF performs the allocation and management of the Resources. It consists of

- base application interfaces (*Port*, *LifeCycle*, *TestableObject*, *PropertySet*, *PortSupplier*, *ResourceFactory*, and *Resource*),
- framework control interfaces (*Application*, *ApplicationFactory*, *DomainManager*, *Device*, *LoadableDevice*, *ExecutableDevice*, *AggregateDevice*, and *DeviceManager*),
- framework services interfaces (*File*, *FileSystem*, *FileManager*, and *Timer*), and
- a *DomainProfile*.

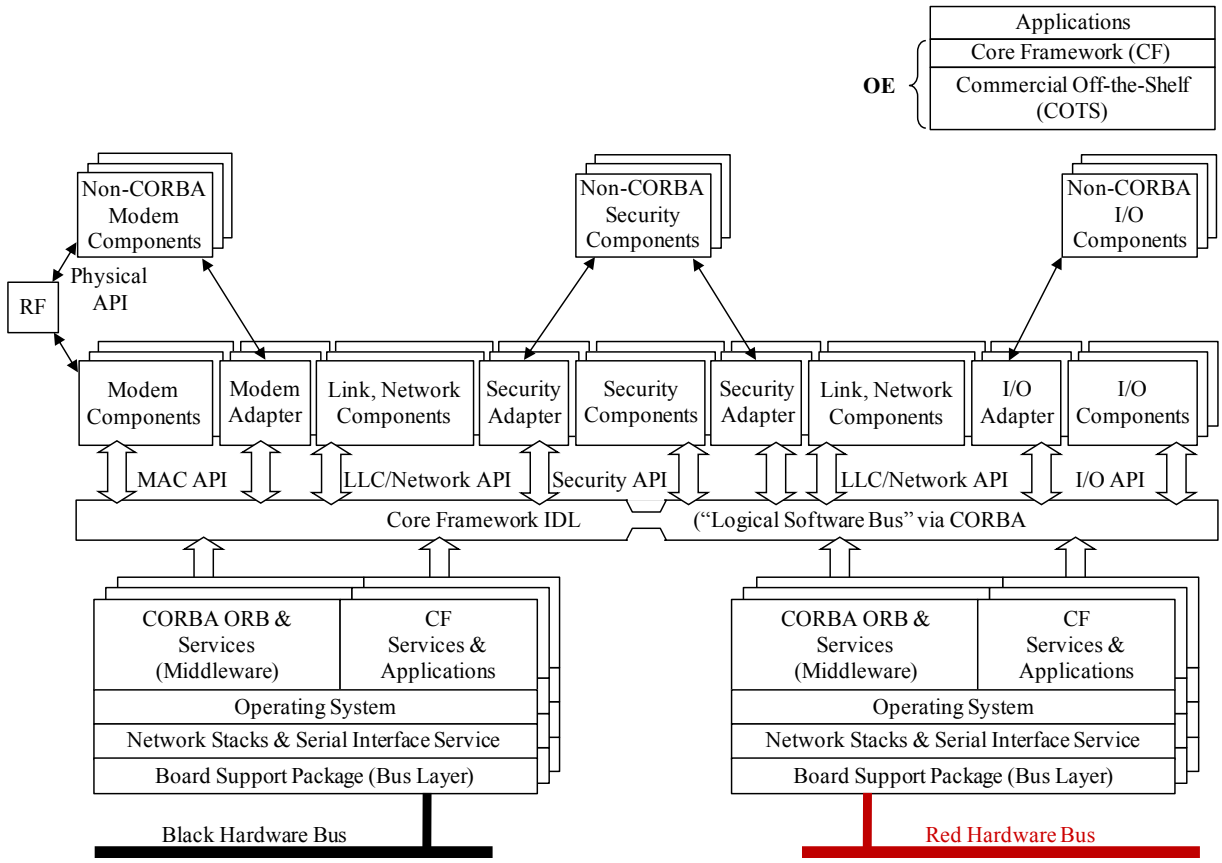


Fig. 2.2. The SCA software framework [88].

The base application interfaces are those that the CF internally uses for controlling the Resources that compose an application. The control interfaces provide control facilities over the SDR system. The service interfaces provide service support to core and non-core applications. The DomainProfile describes the properties of hardware devices and software components [88].

C) Computing Resource Management Facilities

The SCA features some computing resource management facilities, including the DomainProfile, which describes the hardware and software capabilities, and the *releaseObject* operation of the application interface for releasing allocated computing resources [88]. The POSIX OS, however, confines the resource management possibilities of the SCA. Since not supported by the POSIX, the SCA specification does not define methods that facilitate the acquisition of the computing resource states, including the occupation of processing resources and the momentary power consumption, or the communication status, such as the bit error rate (BER) and noise power density.

The SCA, though managing the CPU resource *time*, relies on the best effort timing policy of the POSIX RTOS. This implies that the framework cannot dynamically assign a certain amount of CPU time to one or another process, complicating the application of software mapping algorithms targeting hard real-time constrained applications. Other related difficulties include the delays suffered by the IDL interfaces [90], the large footprints of existing SCA implementations [91], [92], and the problem of dynamically deploying application components [93].

2.2.2 Q-SCA

The Q-SCA is an SDR execution environment with QoS capabilities [86]. It introduces a resource modeling, which specifies the processing, data flow, and latency requirements of waveforms, and adds an admission controller and resource allocator to the SCA framework (Fig. 2.3).

A) Waveform Modeling and Mapping

The Q-SCA waveform model is based on the synchronous data flow (SDF) model, where the data flow requirements are given at compile time [34]. It assumes that processes communicate through unidirectional first-in-first-out (FIFO) channels. The Q-SCA framework models a waveform as a directed acyclic graph (DAG).

A process is characterized by its worst-case execution time C_i, its execution period T_i, and the maximum latency requirement D_i. The unit of C_i is a function of the device type where the process will eventually execute. After determining the particular target processor, the worst-case execution time is converted to actual time units. T_i is specific to input and D_i to output processes.

The directed edge e_{ij} denotes a communication channel from process τ_i to process τ_j. A process cannot start before having received all input data. Edges have two attributes: The number of tokens that τ_i produces and τ_j consumes per invocation. The total number of produced and consumed tokens per invocation needs to match.

Based on this model, application programmers can specify three types of QoS constraints: worst-case execution time, execution period, and maximum latency. To satisfy a waveform's maximum latency requirement─the actual latency is computed as the sum of the processes' response times─waveforms need to receive enough CPU time and memory, among others. These resources are provided by the loadable and executable devices, which map to a set of processing elements.

Each processing element is associated with some matching and allocatable properties. Examples of matching properties are the name and version of the OS. Waveform software components need to comply with the matching properties, or constraints, of the processing elements they deploy. Allocatable properties are the amount of resources that can be provided to a process. The worst-case number of floating point operations per second or the maximum propagation delay would be an allocatable property [86].

B) Q-SCA Core Framework

The Q-SCA supports the modeling and mapping of QoS-constrained waveforms. It, therefore, extends JTRS-SCA domain profiles to allow for QoS and resource specifications, adds services that provide admission control and resource allocation (Fig. 2.3), and extends the software communication bus.

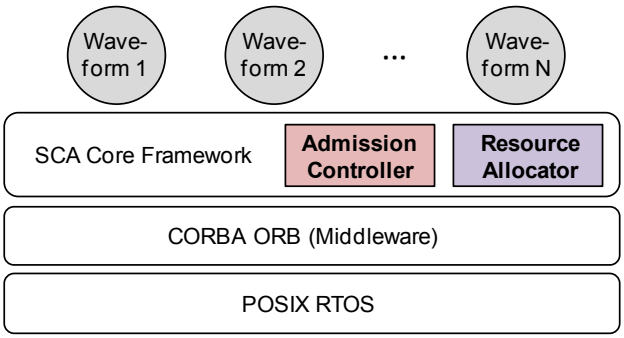


Fig. 2.3. Q-SCA components within the SCA software stack.

The SCA's software assembly descriptor (SAD) is a dedicated XML descriptor that specifies the application structure and components. The Q-SCA extends various fields in the SAD so that the application developers can specify the QoS-related information. The software package descriptor (SPD) and the software component descriptor (SCD) are XML files that specify the processing elements, matching properties, expected computing resource requirements, and the number of produced or consumed tokens for each Port. Developers need to implement a predefined set of configurable property operations that the CF invokes for the resource allocation.

The *ResourceAllocator* keeps track of the available resources through the *deviceCapacities* attribute. Upon a request for creating an application, it checks the schedulability of the application and assigns a loadable or executable device to each application's component. The *createAssignments* and *releaseAssignments* operations specify the resource allocations and deallocations, which the ApplicationFactory executes. The RT-CORBA scheduling service enforces these resource allocations.

C) Computing Resource Management Facilities

The Q-SCA waveform model and the corresponding CF facilitate meeting the QoS constraints of waveforms. Initial results indicate tolerable timing overheads of the SCA-based SDR framework [86].

Some of the Q-SCA concepts will also be found in this dissertation, which introduces a comprehensive SDR computing resource management framework. The Q-SCA and the P-HAL-OE are two alternative SDR frameworks with computing resource management capabilities. They were independently planned and designed. Although not examined in this dissertation, their research results and conclusions may be of mutual interests.

2.2.3 P-HAL-OE

The *platform and hardware abstraction layer* (PHAL) is the SDR framework proposed by the Radio Communications Research Group of the Polytechnic University of Catalonia. PHAL evolved to the P-HAL-OE [94], which we describe here.

A) Design Goals and Challenges

The process of defining a common framework for developing and deploying SDR applications requires eliminating any platform (hardware and supporting software) dependencies. The framework needs to be capable of allocating as many computing resources as required to the applications. This implies the capability for adding and removing hardware components from different providers. Different hardware topologies, configurations, and task assignments impose restrictions on the integration of heterogeneous hardware for creating SDR platforms, requiring a multiplatform abstraction layer and execution environment.

The framework's operational context also needs to be taken into account. Signal processing blocks, in this case, receive input signals which they process to generate output signals. The first design step would then be providing mechanisms for dealing with input and output signals; either as software or hardware modules abstracted from the real implementation. The SDR applications are assembled at the execution time, unifying the corresponding signal processing blocks or objects. The P-HAL-OE should, moreover, support:

1. execution control,

2. hide or abstract platform heterogeneities,
3. data packet instead of processor-specific messaging,
4. online monitoring of execution parameters,
5. computing resource management, and
6. auto-learning or cognitive capabilities for internal resource management.

The first four features in the list have already been implemented, whereas the remaining two are addressed in this dissertation, which provides the theoretical and practical framework for (a cognitive) computing resource management.

B) Principal Characteristics

A hardware abstraction layer (HAL) hides the hardware peculiarities from the software. A HAL can be understood as a software layer, which typically resides within an OS. It ensures that all hardware procedures, such as interprocessor communication, input and output interfaces, memory ranges, and context switches, are completely hidden from the software.

For practical purposes, a HAL eases the deployment of applications that require a direct hardware access. For example, accessing a resource that is located outside a processor's bus requires specific code that deals with the bridging device. HAL routines provide this feature, making the real access to this resource transparent to the applications. Applications can then be designed for any hardware.

The HAL concept is very useful for the execution of SDR applications on different platforms. Hardware transparency is, particularly, desired for easing the design and deployment of SDR applications. On the other hand, heavy signal processing routines with hard real-time constraints require minimal time and processing overheads for maximum efficiency. Hence, P-HAL-OE features only one level of hardware abstractions.

Fig. 2.4 shows a schematic diagram of the layered P-HAL-OE design. The topmost layer is the abstract application layer. It specifies the data flows between the objects of an SDR application. These objects interact with the *PHAL platform* rather than directly with one another. The *PHAL layer*, consisting of the middleware and the platform software layers, hides the peculiarities of the available PE and their physical interconnections, exposing a single virtual processing platform (*PHAL platform*) to the application layers.

Apart from providing a lightweight abstraction layer, the P-HAL-OE provides services that support the execution control of SDR applications and the management of computing resources. These are:

1. real-time information exchange between platforms,
2. isochronisms of data and processes running on different platforms,
3. scheduling of processes on all PEs and execution control,
4. monitoring of state variables and computing resources for generating real-time statistics,
5. distributed management of real-time statistics,
6. parameter initializations and run-time adaptations,
7. homogenous procedure for loading and managing executable files, and
8. automatic network discovery and plug-and-play support.

The P-HAL-OE features a set of components and libraries that facilitate the execution on heterogeneous multiprocessor platforms. These components or libraries can be categorized as either hardware dependent or hardware independent, implementing tasks that are platform-specific and those that are not. The larger the number of hardware independent functions, the easier the portability. Therefore, the platform-specific software provides only elementary services, which can often be implemented with a low software depth.

Fig. 2.5 shows the implementation components of the P-HAL-OE. The application, represented by a single object, can access several services provided by the *PHAL software library*, which is a platform independent library. The *PHAL software daemons* implement these services. They are a set of stand-alone

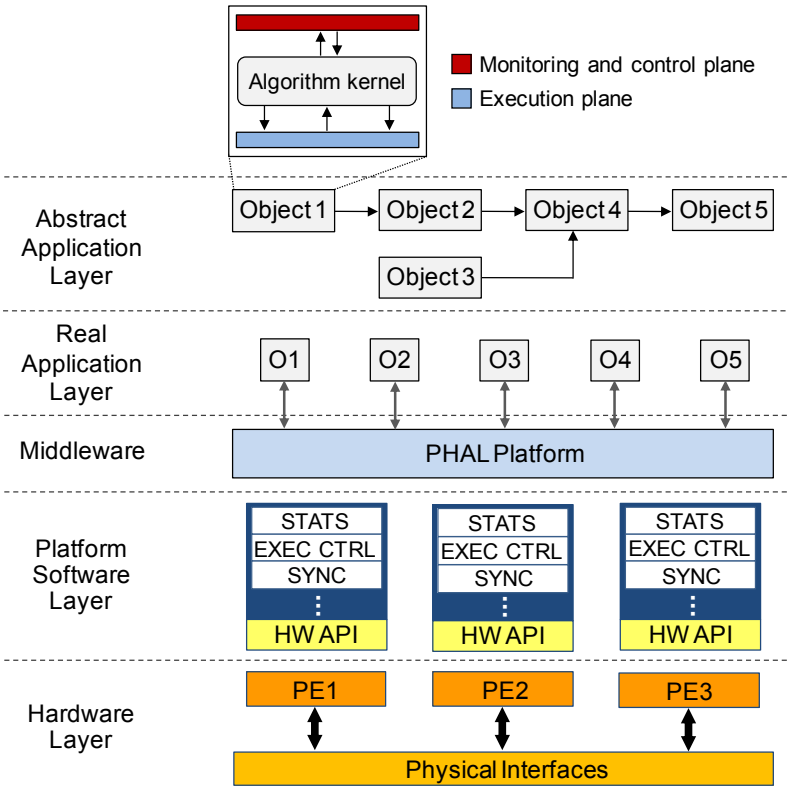


Fig. 2.4. The logical layers of the P-HAL-OE.

entities, which execute only a few operations. They are equivalent to the DomainManager, DeviceManager, ApplicationFactory, and FileManager of the SCA. The following list sketches their principal functions; reference [94] contains further details.

- *CMD MAN*: centralizes all interactions between the higher level control applications and the P-HAL-OE.
- *HW MAN*: performs computing resource management tasks, assigning software requirements to computing resources.
- *SW MAN*: administrates the application and component repositories and definitions.
- *STATS MAN*: provides the object initialization parameters and facilitates monitoring and modifying application variables during execution.
- *BRIDGE*: provides the links for interprocessor data transfers.
- *SYNC MAST*: provides the time reference.
- *FRONT END*: routes the P-HAL-OE control packets between daemons and gathers hardware status information.
- *SW LOAD*: assigns local resources for loading software components and defines the internal data interfaces between them.
- *EXEC CTRL*: controls the real-time execution of software components.
- *STATS*: retrieves and modifies application variables.
- *SYNC*: synchronizes the local times with the time reference.

The platform independent software modules interact with the hardware using the platform-specific *PHAL hardware library*. This library enables the previously stated portability of PHAL functions. The implementation of the hardware library may, in some cases, require the use of an OS. The hardware and software libraries and their application programming interfaces (APIs) are described in [94].

C) Resource Management Facilities

A major design goal for the P-HAL-OE was providing computing resource management facilities, such as

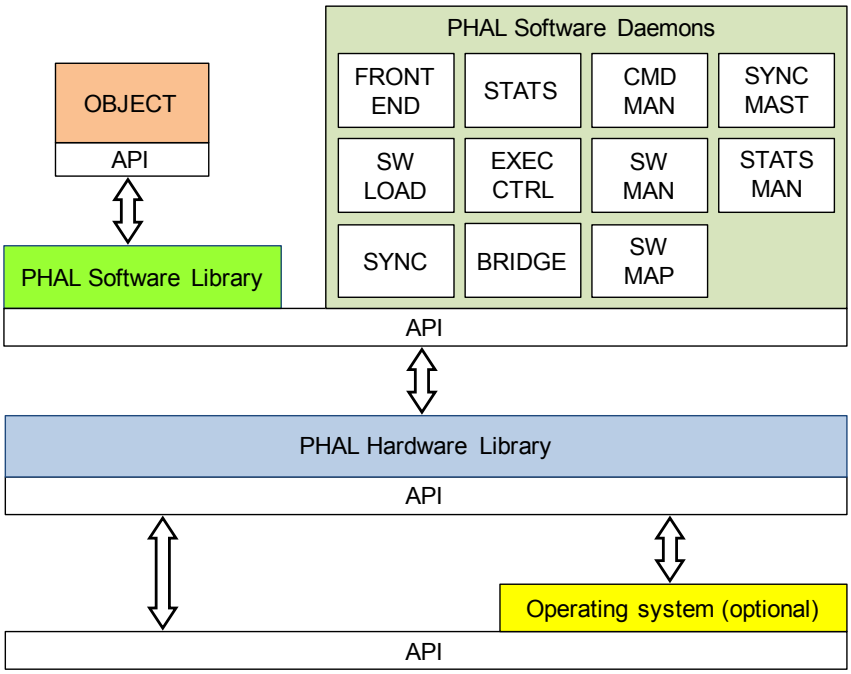


Fig. 2.5. Implementation components of the P-HAL-OE [94].

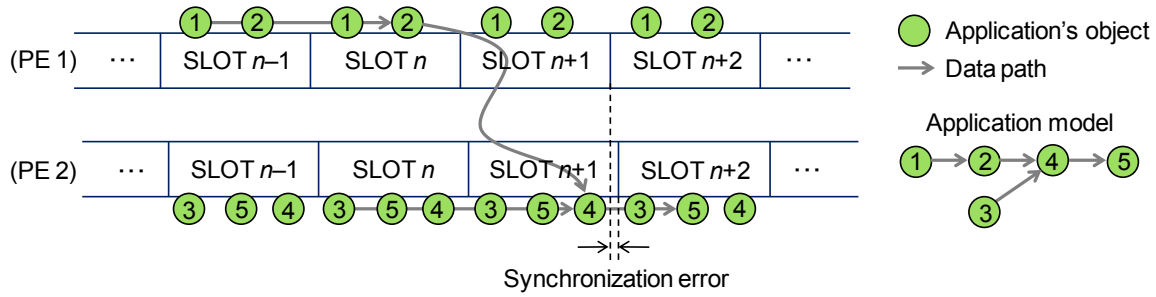


Fig. 2.6. The P-HAL-OE time slots.

- resource awareness (number of sub-platforms, their interconnectivity, etc.),
- time management,
- execution control,
- continuous monitoring of parameters, and
- coherent modeling of the available hardware and software modules.

This dissertation elaborates the corresponding computing resource management framework, which relies on the time management approach described in continuation.

The P-HAL-OE splits the continuous computing time into discrete time slots. Objects are then executed in a pipelined fashion, where data that are produced in one time slot are not consumed before the beginning of the next time slot (Fig. 2.6). This facilitates synchronizing the distributed data processing and ensuring deterministic computing delays through proper mapping and scheduling. The clock synchronization errors, though, need to be within a certain limit, specified as a fraction of the time slot duration. The PHAL platform therefore provides a unique and reliable virtual time. The periodic synchronization process is described in [94].

2.2.4 Other SDR Frameworks

A) Tsao's SDR Software Framework

Tsao et al. [95] introduce a non-CORBA based software framework for SDRs. They propose extending an OS with features that permit the run-time reconfiguration of waveforms. Their SDR framework is divided

into six logical components, including the SDR hardware abstraction layer (SDR_HAL) and the SDR hardware manager (SDR_HM), that provide several functionalities, such as the boot-up and shutdown of an SDR system and its configuration or mode switch. The SDR_HAL decouples the software from the hardware; thus permitting their independent design. The SDR_HM manages the hardware modules, which represent programs that run on hardware devices. The SDR_HM can insert, delete, modify, and configure these modules. Reference [95] also presents several protocols for software and hardware configurations and a prototype implementation, but does not discuss any computing resource management issues.

B) KUAR

The Kansas University Agile Radio (KUAR) is an SDR framework or platform designed for education and research [96]. Its mission is to enable research on wireless networking, communication systems, and spectrum management. KUAR operates within the 5-6 GHz band. It is capable of implementing various modulation algorithms and media access protocols. The hardware architecture consists of a power supply, a control processor (CP), a digital board, including an FPGA, ADCs, DACs, and external interfaces, an RF transceiver, and antennas. The CP can run GNU radio software. KUAR is, except for the antennas, embedded in a shielded box of portable size and can run on battery supply.

The KUAR software architecture consist of radio modules and libraries, control and management programs, drivers, signal processing modules, network protocol stacks, and user applications. This architecture, once fully implemented, may facilitate the design and testing of new waveforms. An automatic or dynamic computing resource management is currently not supported; tools that facilitate and automate robust SDR design and implementation are neither available.

C) GNU Radio

GNU radio is not an SDR framework in the sense that it does provide an execution environment for SDR applications. It is, rather, a free software development toolkit that provides signal processing blocks to implement SDRs using readily-available and low-cost external RF hardware and commodity processors. It supports wireless communications research and implementations [97].

Applications are primarily written using the Python programming language. The performance-critical signal processing path is implemented in C++. The developer is able to implement real-time and high-throughput radio systems using a simple application development environment.

GNU radio is hardware independent. The signal processing blocks are executable on GPPs, although the preferred solution is the Universal Software Radio Peripheral (USRP). An USRP motherboard includes ADCs, DACs, an FPGA, and a programmable USB 2.0 controller. Each fully populated USRP motherboard supports four daughterboards. RF front ends are implemented on the daughterboards. A variety of daughterboards is available for different frequency bands. The USRP's external connection is the USB port, whereas the USRP2 comes with a gigabit Ethernet interface.

2.3 SDR Computing Resource Management Problem

2.3.1 SDR Computing Environment

Modern wireless communications present a hard real-time computing problem [17]. Many computing resources, including processing powers [17] and memory capacities [11], are required for preparing data for the transmission over the unpredictable and error-prone radio channel and even more computing resources are needed for receiving the desired signals, correcting errors, and extracting the embedded information. This explains the traditional, hardware-centric implementations of radio transceivers using processing and power efficient ASICs.

ASICs are, however, inappropriate for SDRs, since providing limited reconfiguration capabilities at most [18]. While a custom integrated circuit can implement entire digital signal processing chains, most reconfigurable or reprogrammable devices do not provide enough processing capacities for implementing more sophisticated radio standards [17], [18]. This explains the need for reconfigurable multiprocessor or multicore platforms, such as clusters of GPPs, DSPs, FPGAs, or a mix of them.

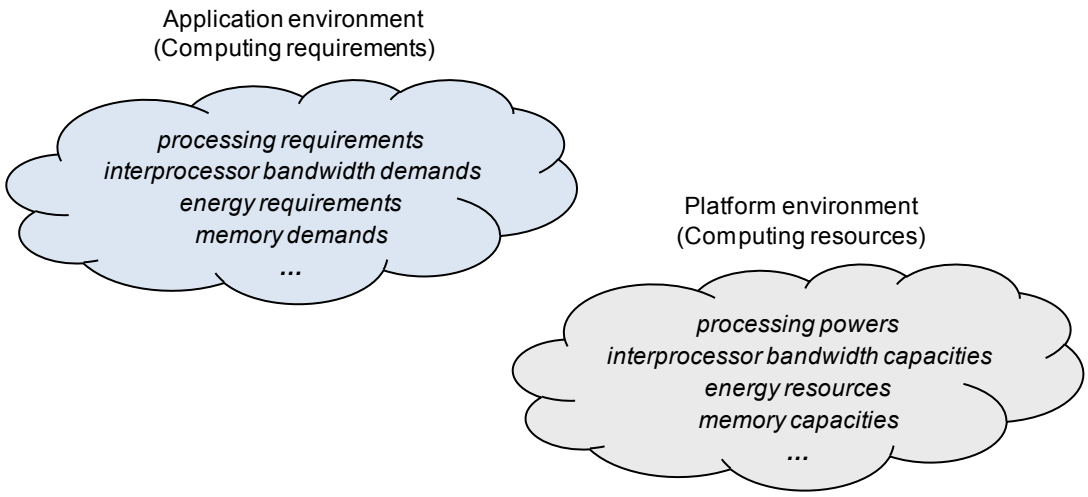


Fig. 2.7. The SDR computing environment.

SDR platforms and applications specify the SDR computing context of this dissertation. The SDR computing environment thus consists of the application and the platform environment (Fig. 2.7). The application environment specifies the computing requirements of SDR applications, whereas the platform environment identifies the computing resources of SDR platforms. Fig. 2.7 indicates some computing resources and the corresponding requirements; these are the processing, interprocessor bandwidth, energy, and memory resources and requirements.

2.3.2 Problem Formulation

The problem consists of defining a flexible computing resource management framework that interfaces the SDR computing resources on one side and the wireless system 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*. Additional constraints, such as energy limitations, are likely and should be accounted for as necessary.

The SDR computing resource management framework should fully support the flexibility of SDR communications. This implies providing support for adding or removing computing resources and facilitating different computing resource management policies with multiple objectives, among others. Computing resource management is required at single-user mobile terminals and multiuser base stations and should, finally, be coordinated with the RRM.

2.3.3 Scope and Assumptions

Fig. 2.8 illustrates the scope of our SDR computing resource management framework. We assume the availability of hardware and software abstractions that hide the details of SDR platforms and applications while providing all relevant information on computing resources and requirements. Based on this information, the framework creates software and hardware computing models, which are the basis for the computing resource management.

Opting for or against hardware abstractions is a tradeoff between programmability, or flexibility, and efficiency. SDR requires a flexible usage of computing resources and, hence, the availability of hardware abstractions. The difficulty lies in finding a unified characterization of computing resources and requirements for conceptually different hardware devices and software programs. In other words, translating device-specific computing resources and tool or software-specific computing requirements to unique metrics requires a profound study, which is beyond the scope of this dissertation. The availability of suitable abstraction layers, though, restrains the applicability of our proposal. Without them, the framework still works for equivalent types of processors and software design tools.

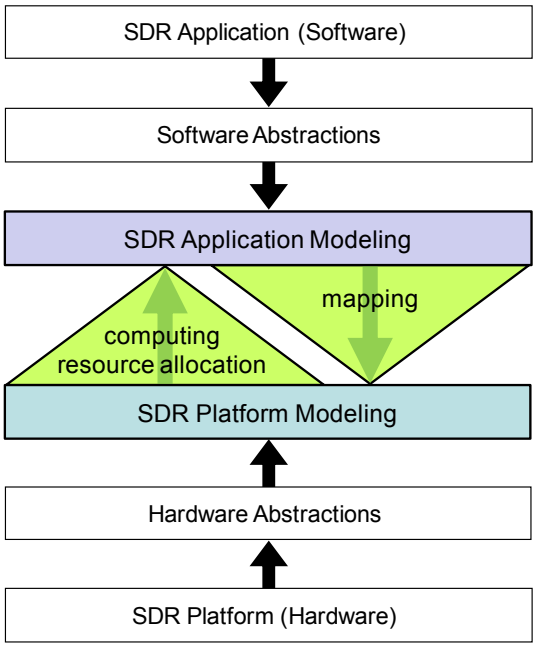


Fig. 2.8. The scope of our SDR computing resource management framework.

Our computing resource management framework assumes the availability of several P-HAL-OE features, including the time management of Section 2.2.3C). Nevertheless, it is not specifically designed for the P-HAL-OE but rather for general applicability. That is, it works within the P-HAL-OE, but could be adapted to any other SDR framework that supports computing resource management.

The P-HAL-OE supplies a pseudo-homogeneous computing environment on top of a heterogeneous computing platform. This includes the already mentioned abstractions of computing resources and requirements. Software implementations and their computing demands for each one of the platform's physical devices are assumed to be available to the P-HAL-OE.

Since our framework should be applicable at SDR-MTs and SDR-BSs and, in principle, be able to handle many different platform and application architectures, we cannot initially assume important simplification. The SDR computing resource management problem of Section 2.3.2 is then essentially a general mapping problem, which is NP-complete [47], [84]. Since finding an optimal solution to an NP-complete problem is generally very time-consuming and computationally impractical [58], we strive for a suboptimal but efficient solution to the SDR computing resource management problem.

Although this dissertation focuses on the radio access or the physical layer of an SDR transceiver, the proposed concepts are general enough for being applied to higher processing layers and network elements as well. The flexibility of our proposal thus permits managing the computing resource of any part of the radio system. It may, moreover, be practical for other (real-time) computing contexts.

2.4 Related Work

As opposed to the relatively few publications that address SDR computing resource management issues (Section 1.4.2A)), heterogeneous computing literature contains a plentitude of contributions to multiprocessor mapping and scheduling. Heterogeneous computing refers to a coordinated use of distributed and heterogeneous computing resources [33]; it is similar to the grid computing [22] or metacomputing [24] concepts. SDR computing belongs to the heterogeneous computing context (Section 1.4.2B)). The SDR computing resource management problem is then a particular heterogeneous computing problem.

Heterogeneous computing research has addressed 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: [34], [35], [36], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [81], for example, aim at minimizing the application's execution time, [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], among others, focus on meeting real-time deadlines, whereas [37], [56], [57], [58], [59], [60], [61], [72], [73],

[74], [75], [76], [77], [78], [84] pursue additional or other objectives. The following subsections depict some of these articles in more detail.

2.4.1 Maximizing Speedup

Many contributions on multiprocessor mapping and scheduling focus on minimizing the schedule length or *makespan*. The schedule length or *makespan* refers to the execution time of an application or set of applications and is usually measured from starting the execution of the first process until finishing the execution of the last. This goal equivalently maximizes the application speedup, defined as the shortest execution time on a single processor (sequential processing) divided by the schedule length on the multiprocessor platform (parallel processing).

Hu's early work [35] assumes unit link costs for interprocessor communications, whereas Bondalapati [52] addresses sequential applications. Other contributions, including [38], [39], [40], [41], assume applications with a certain degree of parallelism (parallel applications) and interprocessor communication overheads. Lee and Aggarwal [38] present an efficient mapping scheme for different objective functions, minimizing the communication overhead for increasing the application speedup. They suggest deriving the mapping strategy as a function of the optimization goal. That is, before developing a suitable mapping scheme, they formulate the objective function.

Selvakumar and Murthy [39] introduce a scheduling algorithm that maps and schedules precedence constrained task graphs with nonnegligible communication requirements while considering contention for the communication channels. Sih and Lee [40] introduce the *dynamic level scheduling* (DLS) for mapping and scheduling precedence-constrained tasks and their data flows to heterogeneous processor architectures with limited or irregular interconnection structures. The dynamic levels are obtained as a function of the momentary state of the processing and communication resources and determine the next task-processor pair to be scheduled.

The generalized mapping strategy of [41] applies graph theoretic, mathematical programming, and heuristics approaches. It begins with a graphical representation of the parallel algorithm (problem graph) and the parallel computer (host graph). The host graph is then extended (extended host graph consisting of pseudo-processors) as a function of the problem graph, which is finally mapped to the extended host graph using a two-step optimization scheme.

Lee and Messerschmitt [34] formalize the scheduling problem for digital signal processing. They discuss the suitability of SDF graphs─a subset of large grain data flow (LGDF) graphs with a priori information about produced and consumed data at each node─for modeling digital signal processing chains. They develop a theory for scheduling SDF programs on single and multiprocessors systems and introduce a *synchronous large grain compiler*. This compiler converts a LGDF description of a signal processing system into a set of sequential programs that run on one or several machines so that the throughput is maximized while using a finite memory amount.

The *dynamic critical-path (DCP)* scheduling algorithm [42] schedules critical nodes as soon as possible. It dynamically determines the critical path of the task graph, selects the next node on that path, and schedules it. Each processor's partial schedule may be dynamically rearranged; that is, the execution order of nodes is not fixed until having scheduled all nodes. Based on the same principle, the *dynamic priority scheduling (DPS)* heuristic [45] dynamically assigns task priorities to avoid scheduling less important tasks before the more important.

Alhusaini et al. [48] present a unified resource scheduling framework for metacomputing systems. The framework considers different types of resources─processing and communication resources as well as data repositories─and permits advanced resource reservations. The system resources are collectively scheduled, minimizing the makespan while preserving advanced reservations. Similarly, [49], [50] address the resource co-allocation problem, where multiple resources need to be allocated to concurrent tasks of various applications. Tasks that share one or more resources cannot be simultaneously executed. Two graphs are used to represent applications: a DAG, which indicates the precedence constraints and data flow requirements, and a *compatibility graph*, which captures the resource sharing constraints. The solution is based on dynamically finding maximal independent sets─the largest set of tasks with no resource sharing constraints─among those tasks whose predecessors' resource requirements have been completely satisfied [49]. The initial, offline schedule plan can be adapted at run time, taking advantage of early releases of allocated resources and runtime variations in computation and communication costs [50].

Kwok et al. [51] consider a scenario where some characteristics of subtasks are unknown a priori and will change during execution. They implement and evaluate a semi-static mapping methodology, which starts with an initial mapping and dynamically decides whether to perform a remapping between the iterations of the application.

2.4.2 Meeting Real-Time Deadlines

Peng et al. [62] and Hou and Shin [63] 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 of [64], where the objective is meeting all real-time deadlines and not minimizing the overall execution time. If a task's deadline cannot be met on a particular processing node, this task can be sent to another node [65]. The task model in [64] or [65] accounts for worst case processing times, deadlines, and resource requirements; no precedence constraints are assumed.

Instead of assuming worst-case application requirements, [66] proposes adapting the resource allocation to face the runtime changes in the application environment. It describes and evaluates models and mechanisms for adaptive resource allocation in the context of embedded high performance applications with real-time constraints. Based on the same principle, [67] presents a mathematical modeling for adaptive resource management. It precisely models fixed hardware─a network of processors─and dynamic, real-time software. It also proposes a framework for allocation algorithms, supporting the three constraints *application-host validity*, *minimum security level*, and *real-time deadlines* while maximizing the overall utility of the system.

Gertphol et al. [69] also address dynamic real-time environments. Instead of relying on adaptive resource allocations, they discuss robust resource allocations for dynamic real-time systems. A robust resource allocation would avoid the need for dynamic reconfigurations due to run-time parameter variations while still meeting the system constraints (deadlines). The paper introduces the *maximum allowable increase in load* (MAIL) metric and proposes a method for determining an allocation that can tolerate a certain amount of load increase.

A list scheduling framework for the run-time stabilization of static tasks with hard real-time deadlines and dynamic tasks with soft real-time deadlines is described in [70]. It allows for static and dynamic task-to-processor allocations for increasing the processor utilization and response time of dynamic workloads. Moreira et al. [71] address hard real-time streaming applications and assume a scenario where jobs enter and leave a homogeneous multiprocessor system at any time during operation. The proposal combines global resource allocation (admission control or mapping) with local resource provisioning (scheduling).

2.4.3 Following Other or Multiple Objectives

Stujik [72] addresses the mapping of streaming applications to multiprocessors, considering multi-rate and cyclic dependencies between tasks and a NoC-based MP-SoC platform. He presents an SDF model for memory accesses, discusses a strategy that binds multiple SDF graphs with throughput constraints to a heterogeneous MP-SoC [73], analyzes the throughput versus storage tradeoff (minimizing the required buffer space while meeting the throughput constraint) [74], and proposes several algorithms for a resource-efficient routing and scheduling of communication paths [75].

Radulescu and van Gemund [79] reduce the complexity of list scheduling algorithms with static and dynamic priorities without sacrificing performance. Kim et al. [76] discuss a performance measurement framework for distributed heterogeneous networks. They introduce the *flexible integrated system capability* (FISC) measure, a multi-dimensional metric that combines a broad range of attributes, such as deadline, security, and application specific QoS attributes. It quantifies the overall value of the performance received by a set of applications in a distributed computing environment.

Darbha and Agrawal [57] propose a *task duplication based scheduling* (TDS) algorithm with low complexity. TDS considers executing certain tasks on various processors for avoiding excessive interprocessor data flows. Bajaj and Agrawal [58] propose the *task duplication-based scheduling algorithm for network of heterogeneous systems* (TANH). Both algorithms are shown to be optimal under certain conditions and each has a complexity of $O(V^2)$, where V captures the number of nodes in the DAG [57], [58]. Bansal et al. [61] propose a TDS algorithm that minimizes the makespan while avoiding redundant dupli-

cations. Their *selective duplication* algorithm is suitable for multiprocessor systems with a limited number of interconnection-constrained processors.

Dogan and Özgüner [60] propose the *reliable dynamic level scheduling algorithm*, which accounts for the execution time as well as the reliability of applications. That is, apart from minimizing the scheduling length, an application's failure probability can also be minimized. Ahmad and Kwok [59] aim at minimizing the makespan while reducing the execution time of the scheduler. They suggest parallelizing the scheduling algorithm for its distributed execution on multiple machines. In [37] they propose a taxonomy for classifying different multiprocessor scheduling solutions and analyze 27 algorithms that maximize the application speedup.

Doulamis et al. [77] discuss the fair sharing of CPU rates for grid computing systems. They suggest allocating resources to users as a function of resource availabilities, user demands, and socioeconomic values. Seven different problems and their solutions are finally discussed in [78]. These include optimal job scheduling techniques ([78], pp. 99-112) and a design methodology that minimizes contention with minimum communication resources ([78], pp. 174-190).

2.5 Conclusions

Many multiprocessor mapping and scheduling problems have been addressed in the heterogeneous computing context. SDR computing is just another heterogeneous computing challenge. The direct application of existing computing resource management approaches is, however, difficult because of the specific SDR computing characteristics and their radio implications. The SDR computing resource management context, particularly, requires flexibility and efficient 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 dynamic radio environment and just have to be met. Speeding up an SDR application is, specifically, not necessary.

SDR requires a (more) general computing resource management framework regarding platforms, waveforms, algorithms, and policies. Nevertheless, concepts such as global mapping, followed by local scheduling [71] as well as robust [69] and adaptive [66], [67] resource allocations, are practical in SDR. The latter concepts facilitate partial reconfigurations, which can dynamically modify or exchange certain signal processing blocks to face (runtime) changes in the application and radio environments. The resource co-allocation problem [49] is also relevant in SDR: All (types of) computing resource constraints need to be satisfied for ensuring real-time execution and appropriate service delivery.

Lee and Messerschmitt [34] found that SDFG accurately model digital signal processing applications, where the sampling rate specifies the data flow requirements at compile time. In the 80s they already applied general-purpose computing techniques for digital signal processing [34]. SDR extends the digital domain of wireless communications; applying general-purpose computing techniques in SDR is then a logical inference.

Most of the currently available SDR frameworks assume fixed or offline computing resource allocations. Only a few research efforts, such as the P-HAL-OE, provide dynamic computing resource management capabilities. The Q-SCA, on the other hand, introduces computing resource management facilities to the SCA. A flexible SDR framework with real-time computing resource management capabilities seems to be the current trend (Fig. 2.9).

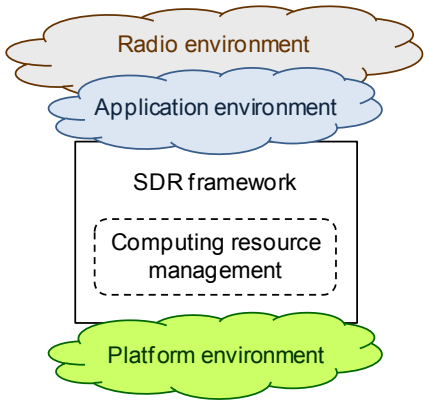


Fig. 2.9. Envisaged SDR framework.

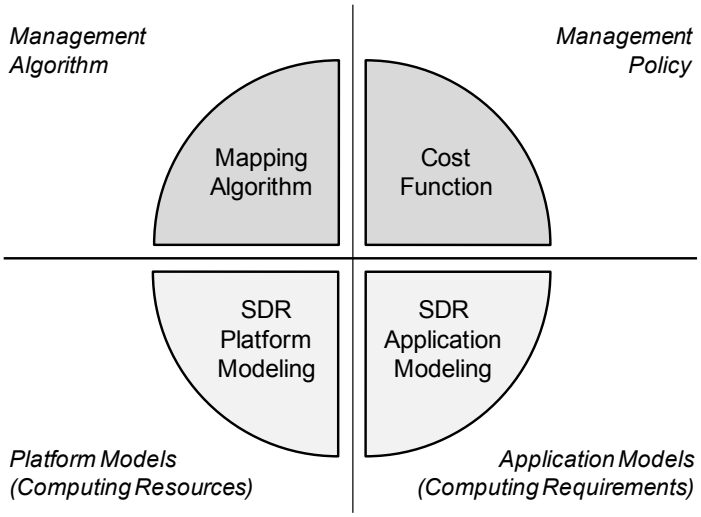


Fig. 2.10. Modular SDR computing resource management framework.

Flexibility is essential for future-proof SDR or cognitive radio execution environments in many senses: platform and application support, parameter management, computing resource management strategies, and so forth. Theoretically unlimited flexibility would facilitate optimally dealing with the dynamic and unpredictable radio and computing environments, taking advantage of the radio and computing diversities of B3G and SDR for offering personalized user services. This implies coordinating the radio with the computing resource management.

The modularity of SDR provides maximum flexibility: Small software modules can easily be developed, updated, exchanged, and so on. Similarly, a modular hardware design permits a flexible system upgrade. We adopt these concepts for designing a modular computing resource management framework. Our framework consists of two principal modules: the *SDR computing system modeling* (Chapter 3) and the *SDR computing resource management* (Chapter 4). We, furthermore, suggest a modular system modeling and management (Fig. 2.10), which eases applying different policies for monitoring and managing all relevant computing resources and requirements.

3

SDR Computing System Modeling

3.1 Introduction

This chapter introduces a modeling that accounts for several SDR-specific system characteristics. Section 3.2 discusses the computing resource management facilities that constitute the basis for the system models of Section 3.3. The platform and application models (Fig. 3.1) capture the relevant SDR platform and application features, including computing resources and requirements. Section 3.4 explains how the SDR computing constraints of Section 2.3 can eventually be met. We exemplify the modeling proposal (Section 3.5) before discussing additional modeling features and possible extensions (Section 3.6).

SDR application and *SDR platform* refer to the software and hardware part of an SDR system. The following definitions formally define the term *SDR application* and its components.

Definition 2 *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 3 *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 4 *An* SDR process *is the smallest manageable unit and symbolizes an indivisible binary code.*

Although our computing resource management framework considers SDR applications and functions, it may work as well on the basis of SDR processes or a mix of SDR functions and processes.

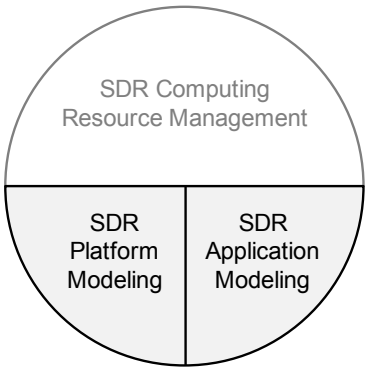


Fig. 3.1. Modular SDR computing system modeling.

3.2 Computing Resource Management Facilities

3.2.1 Metrics

An SDR platform represents an SDR-MT or SDR-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 MP-SoC performances are typically given in million instructions per second (MIPS). Because of the differing resource and execution concepts of different types of processors (parallel versus sequential processing), we generally consider the available silicon area and time as the fundamental processing resources for executing SDR applications or parts of them.

Related work considers processing powers and interprocessor bandwidth capacities as the principal computing resources (Section 2.4). Hence, the amount of processing and interprocessor bandwidth resources abstracts an SDR platform here. Mitola [3] proposed characterizing all platform features, including the processing powers and bandwidths, in equivalent million operations per second (MOPS). We adopt this unit for characterizing the processing resources, assuming that abstraction layers facilitate a homogeneous characterization of heterogeneous devices (Section 2.3.3). Similarly, we quantify any interprocessor communication facilities 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, bit precision, optimization level (speed versus memory or area), and performance demand. The data flow requirements are, basically, a function of the bit precision and the sampling rate. Despite these dependencies, we use the same metrics for characterizing the computing requirements of SDR applications. The implicit timing requirements need to be specified as a function of the radio link timing constraints, as discussed in Sections 3.2.2 and 3.4.

3.2.2 Time Slots and Pipelining

We consider processing time as just another limited computing resource. MOPS and Mbps embed this critical resource and thus permit an implicit time management. Here we discuss two mechanisms that ease the SDR computing resource management. These mechanisms are supported by the P-HAL-OE.

Data that are transmitted or received over the wireless link need to be processed for as long as there are data to transmit or receive. An SDR application will execute during the entire user session or part of it, even though there might be periods where no user data are 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

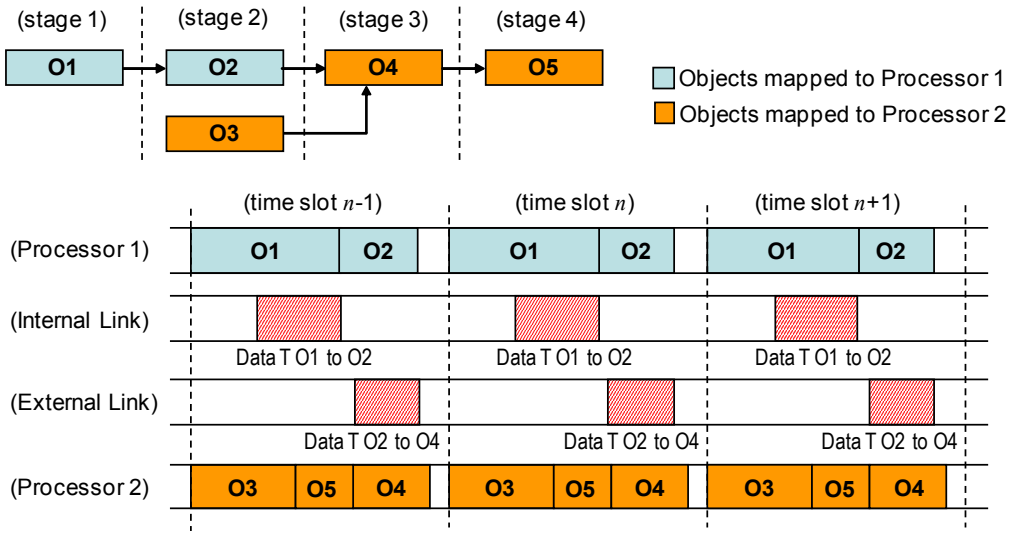


Fig. 3.2. Illustration of the time slot division and pipelining.

resource *time* in equidistant computing time slots and the SDR application in pipelining stages. Fig. 3.2 illustrates this.

The pipelined execution of an SDR application establishes that, in any computing 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. 3.2). This introduces synchronization requirements, which P-HAL-OE can satisfy. Pipelining also introduces latency, which must be maintained within the radio service and QoS-dependent limits. Section 3.4 explains how to achieve this.

The introduction of the computing time slot, time slot from here on, allows identifying a processor's computing capacity on time slot basis. This provides the basic mechanism for an efficient computing resource management. We, therefore, introduce the new units *million operations per time slot* (MOPTS) and *megabits per time slot* (MBPTS) as $t_{TS} \cdot$ MOPS and $t_{TS} \cdot$ Mbps, where t_{TS} is the time slot duration, which is specified in Section 3.4. MOPTS and MBPTS synchronize the available computing resources with the time slot management and are the basic units of the modeling proposal that follows.

3.3 Modeling

Based on the above facilities, we introduce a modeling that consists of two parts, the platform modeling and the application modeling (Fig. 3.1). The platform modeling characterizes SDR platforms and their computing resources (Section 3.3.1), whereas the application modeling abstracts SDR applications and their computing requirements (Section 3.3.2). Our proposal, particularly, features general platform and application templates, instances of these templates─platform and application models─and a set of parameters derived from these models. We use the symbols of Table 3.1, cursive symbols indicating numerical variables and cursive and bold expressions identifying matrices.

3.3.1 Platform Modeling

A) Platform Templates

We model the platform features of SDR platform D through the general platform template

$$(\boldsymbol{R}_{\mathrm{P}}{}^{t'})^{\boldsymbol{D}} = \begin{pmatrix} ({R_{\mathrm{P}}}^{t'}|_{11})^D & ({R_{\mathrm{P}}}^{t'}|_{12})^D & \cdots & ({R_{\mathrm{P}}}^{t'}|_{1,Y(t')^D})^D \\ ({R_{\mathrm{P}}}^{t'}|_{21})^D & ({R_{\mathrm{P}}}^{t'}|_{22})^D & \cdots & ({R_{\mathrm{P}}}^{t'}|_{2,Y(t')^D})^D \\ \vdots & \vdots & \ddots & \vdots \\ ({R_{\mathrm{P}}}^{t'}|_{X(t')^D,1})^D & ({R_{\mathrm{P}}}^{t'}|_{X(t')^D,2})^D & \cdots & ({R_{\mathrm{P}}}^{t'}|_{X(t')^D,Y(t')^D})^D \end{pmatrix}, \tag{3.1}$$

where t' indicates the platform feature (Table 3.1). This template serves for capturing different platform characteristics, including computing architecture and resources. It is therefore without unit. We also introduce

$$(\boldsymbol{R}^t)^{\boldsymbol{D}} = \begin{pmatrix} (R^t_{11})^D & (R^t_{12})^D & \cdots & (R^t_{1,Y(t)^D})^D \\ (R^t_{21})^D & (R^t_{22})^D & \cdots & (R^t_{2,Y(t)^D})^D \\ \vdots & \vdots & \ddots & \vdots \\ (R^t_{X(t)^D,1})^D & (R^t_{X(t)^D,2})^D & \cdots & (R^t_{X(t)^D,Y(t)^D})^D \end{pmatrix}, \tag{3.2}$$

which represents the specific template for modeling the computing resources of SDR platforms, where $t \in 1, 2, \ldots, T$ specifies the computing resource type. A computing resource is a platform feature; that is, $(\boldsymbol{R}^t)^{\boldsymbol{D}} \subseteq (\boldsymbol{R}_{\mathrm{P}}{}^{t'})^{\boldsymbol{D}}$.

B) Platform Models

The processing and bandwidth resources can be modeled as instances of (3.1) and (3.2). The *device model*

$$(\boldsymbol{R}_{\mathrm{P}}{}^{1})^{\boldsymbol{D}} = (\boldsymbol{R}^{1})^{\boldsymbol{D}} = \boldsymbol{C}^{\boldsymbol{D}} = ((C_1)^D, (C_2)^D, \ldots, (C_{N(D)})^D) \text{ MOPTS} \tag{3.3}$$

Table 3.1 Modeling symbols.

Symbol	Range [unit]	Description
D	I, II, III, IV, …	SDR platform index
d	i[1], ii, iii, iv, …	SDR application index
$N(D)$	$\mathbb{N}$ (natural numbers without 0)	number of processors on SDR platform D
$M(d)$	$\mathbb{N}$	number of SDR functions of SDR application d
$(P_j)^D$	$(P_1)^D$, $(P_2)^D$, …, $(P_{N(D)})^D$	processor on SDR platform D
$(f_i)^d$	$(f_1)^d$, $(f_2)^d$, …, $(f_{M(d)})^d$	SDR function of SDR application d
T	$\mathbb{N}$	number of modeled computing resource types
t	1, 2, …, T	computing resource type index
T'	$\mathbb{N}$	number of modeled platform features (including computing resources)
t'	1, 2, …, T'	platform feature index
T''	$\mathbb{N}$	number of modeled application features (including computing requirements)
t''	1, 2, …, T''	application feature index
$(\boldsymbol{R}_\mathrm{P}{}^{t'})^{\boldsymbol{D}}$		general platform template
$(\boldsymbol{R}_\mathrm{A}{}^{t''})^{\boldsymbol{d}}$		general application template
$(\boldsymbol{R}^{t})^{\boldsymbol{D}}$		computing resource template; $(\boldsymbol{R}^{t})^{\boldsymbol{D}} \subseteq (\boldsymbol{R}_\mathrm{P}{}^{t'})^{\boldsymbol{D}}$
$(\boldsymbol{r}^{t})^{\boldsymbol{d}}$		computing requirement template; $(\boldsymbol{r}^{t})^{\boldsymbol{D}} \subseteq (\boldsymbol{R}_\mathrm{A}{}^{t''})^{\boldsymbol{D}}$
$X(t)^D$ and $X(t')^D$	$\mathbb{N}$	number of rows in $(\boldsymbol{R}^{t})^{\boldsymbol{D}}$ and $(\boldsymbol{R}_\mathrm{P}{}^{t'})^{\boldsymbol{D}}$
$Y(t)^D$ and $Y(t')^D$	$\mathbb{N}$	number of columns in $(\boldsymbol{R}^{t})^{\boldsymbol{D}}$ and $(\boldsymbol{R}_\mathrm{P}{}^{t'})^{\boldsymbol{D}}$
$x(t)^d$ and $x(t'')^d$	$\mathbb{N}$	number of rows in $(\boldsymbol{r}^{t})^{\boldsymbol{d}}$ and $(\boldsymbol{R}_\mathrm{A}{}^{t''})^{\boldsymbol{d}}$
$y(t)^d$ and $y(t'')^d$	$\mathbb{N}$	number of columns in $(\boldsymbol{r}^{t})^{\boldsymbol{d}}$ and $(\boldsymbol{R}_\mathrm{A}{}^{t''})^{\boldsymbol{d}}$
u, v	$\mathbb{N}$	unspecific indices
n	$\mathbb{N}$	unspecific natural number
t_TS	$\mathbb{R}^+$ (nonnegative real numbers) [s]	time slot duration
$(t_\mathrm{TS})^d$	$\mathbb{R}^+$ [s]	time slot duration for SDR application d
$(n_\mathrm{TS})^d$	$\mathbb{N}$	number of pipelining stages of SDR application d
$(L_\mathrm{MAX})^d$	$\mathbb{R}^+$ [s]	maximum allowed latency for SDR application d

[1] *Descriptive labels are also used.*

absorbs the processing powers of processors P_1 to $P_{N(D)}$ of SDR platform D ($(C_u)^D \in \mathbb{R}^+$). Without loss of generality, devices are labeled in order of decreasing processing capacities; that is $(C_1)^D \geq (C_2)^D \geq (C_3)^D \geq \ldots \geq (C_{N(D)})^D$.

Matrix

$$(\boldsymbol{R}_\mathrm{P}{}^{2})^{\boldsymbol{D}} = (\boldsymbol{R}^{2})^{\boldsymbol{D}} = \boldsymbol{L}^{\boldsymbol{D}} = \begin{pmatrix} (L_{11})^D & (L_{12})^D & \cdots & (L_{1,N(D)})^D \\ (L_{21})^D & (L_{22})^D & \cdots & (L_{2,N(D)})^D \\ \vdots & \vdots & \ddots & \vdots \\ (L_{N(D),1}) & (L_{N(D),2})^D & \cdots & (L_{N(D),N(D)})^D \end{pmatrix} \text{MBPTS} \tag{3.4}$$

describes the communication resources of platform D ($(L_{uv})^D \in \mathbb{R}^+$). $(L_{32})^D$, for instance, specifies the available bandwidth per time slot of the directed communication link between $(P_3)^D$ and $(P_2)^D$. In other words, $(L_{32})^D$ is the bandwidth capacity that is available for the directed data transfer from the local data memory of processor $(P_3)^D$ to the local data memory of processor $(P_2)^D$. If there is more than one directed physical link between $(P_3)^D$ and $(P_2)^D$, $(L_{32})^D$ captures the accumulated bandwidth resources. $(L_{uu})^D$ indicates the available bandwidth for intra-processor data movements. Intra-processor bandwidth capacities can be different for different devices.

$\boldsymbol{L}^{\boldsymbol{D}}$ is based on the concept of the *adjacency matrix*, which holds a pair of 1s on each existing undirected link and 0s, otherwise [106]. It informs about the communication topology (interconnectivity network) and the communication resources (bandwidths), assuming a network that consists of unidirectional

communication lines between each pair of processors. Additional information is needed for modeling shared links. We therefore suggest a more general communication modeling that distinguishes between the communication topology (3.5) and the communication resources (3.6).

$$(\boldsymbol{R}_{\mathrm{P}}^{3})^{\boldsymbol{D}} = \boldsymbol{I}^{\boldsymbol{D}} = \begin{pmatrix} (I_{11})^D & (I_{12})^D & \cdots & (I_{1,N(D)})^D \\ (I_{21})^D & (I_{22})^D & \cdots & (I_{2,N(D)})^D \\ \vdots & \vdots & \ddots & \vdots \\ (I_{N(D),1})^D & (I_{N(D),2})^D & \cdots & (I_{N(D),N(D)})^D \end{pmatrix} \tag{3.5}$$

represents the logical interconnection model, where $(I_{uv})^D \in 1, 2, \ldots, N(D) \cdot N(D)$ can be viewed as the numerical label of the logical link between $(P_u)^D$ and $(P_v)^D$. A logical link corresponds to a directed, or unidirectional, communication line between a pair of processors. These logical links map to physical links. $\boldsymbol{I}^{\boldsymbol{D}}$ thus points to entries in

$$(\boldsymbol{R}_{\mathrm{P}}^{4})^{\boldsymbol{D}} = (\boldsymbol{R}^{3})^{\boldsymbol{D}} = \boldsymbol{B}^{\boldsymbol{D}} = ((B_1)^D, (B_2)^D, \ldots, (B_{N(D)\cdot N(D)})^D)\ \text{MBPTS}, \tag{3.6}$$

which captures the physical link resources $((B_u)^D \in \mathbb{R}^+)$. $(B_u)^D$, where $u = (I_{32})^D$ for instance, is the maximum bandwidth that is available for the directed data transfer from the local data memory of processor $(P_3)^D$ to the local data memory of processor $(P_2)^D$. It would be zero if the physical link is unavailable or nonexistent. The first $N(D)$ elements of $\boldsymbol{B}^{\boldsymbol{D}}$, $(B_1)^D$ to $(B_{N(D)})^D$, capture the processor-internal communication resources of processor $(P_1)^D$ to $(P_{N(D)})^D$; hence, $(I_{uu})^D \in 1, 2, \ldots, N(D)$. Since processor-internal data movements are typically orders of magnitude faster than processor-external data transfers, we can label logical links so that $(B_1)^D \geq (B_2)^D \geq (B_3)^D \geq \ldots \geq (B_{N(D)\cdot N(D)})^D$. Unused elements in (3.6) are filled with 0s.

This document considers that a dedicated physical link establishes an exclusive connection for the (unidirectional or bidirectional) communication between a processor pair, whereas a shared physical link is accessible by more than two processors. Full-duplex (FD) communication here refers to a dedicated communication channel for each data flow direction between two or more processors (independent of the actual bandwidth, which may be zero in one data flow direction). Half-duplex (HD), on the other hand, indicates a bidirectional link, which needs to be time-shared between the data flow in one and the other direction. That is, FD and HD distinguish between unidirectional and bidirectional physical links.

The *communication models* (3.5) and (3.6), as opposed to (3.4), facilitate modeling different types of interprocessor communication links and networks: dedicated or shared and uni or bidirectional. For example, all entries in $\boldsymbol{I}^{\boldsymbol{D}}$ pointing to different entries in $\boldsymbol{B}^{\boldsymbol{D}}$ would indicate an interconnection network that consists of dedicated and unidirectional communication lines. ($\boldsymbol{L}^{\boldsymbol{D}}$ is equivalently applicable in this case.) If each of the $N(D) \cdot N(D)$ elements of $\boldsymbol{B}^{\boldsymbol{D}}$ is, furthermore, greater than 0—fully connected platform—the logical interconnection model matches the physical interconnection network. At the other extreme, $(I_{uv})^D = N(D) + 1$ for all $u \neq v$ indicates a platform that connects all its processors through a bidirectional shared bus of bandwidth $(B_{N(D)+1})^D$. Section 3.5 illustrates some examples.

We assume direct memory access (DMA) or pointer transfers for processor-internal data flows. Since these techniques facilitate very fast data transfers, processor-internal bandwidths can be modeled as if they were infinite [81]; that is, $(L_{11})^D = (L_{22})^D = \ldots = (L_{N(D),N(D)})^D = \infty$ and $B_1 = B_2 = \ldots = B_{N(D)} = \infty$. Nevertheless, the general computing resource modeling and management approaches of this dissertation permit (processor-internal) bandwidth adjustments in later research stages.

Table 3.2 Platform parameters, part I: computation.

Expression	Range & Unit	Denomination
$(C_{\mathrm{T}})^D = (C_1)^D + (C_2)^D + \ldots + (C_{N(D)})^D$	$[0, \infty)$ MOPTS	Total processing capacity
$(C_{\mathrm{M}})^D = \dfrac{(C_{\mathrm{T}})^D}{N(D)}$	$[0, \infty)$ MOPTS	Mean processing capacity
$(H_{\mathrm{C}})^D = \begin{cases} \dfrac{2}{(C_{\mathrm{T}})^D} \cdot \sqrt{\dfrac{1}{N(D)} \cdot \sum_{u=1}^{N(D)} ((C_u)^D - (C_{\mathrm{M}})^D)^2}, & \text{if } (C_{\mathrm{T}})^D > 0 \\ 0, & \text{otherwise} \end{cases}$	[0, 1]	Processing resource heterogeneity

Table 3.3 Platform parameters, part IIa: communication (based on $\boldsymbol{L}^{D}$).

Expression	Range & Unit	Denomination
$CON^D = \begin{cases} \dfrac{\sum_{u=1}^{N(D)} \sum_{v=1,v\neq u}^{N(D)} sign\{(L_{uv})^D\}}{N(D)^2 - N(D)}, & \text{if } N(D) > 1 \\ 0, & \text{otherwise} \end{cases}$	[0, 1]	Platform's connectivity
$(B_{\mathrm{T}})^D = \sum_{u=1}^{N(D)} \sum_{v=1,v\neq u}^{N(D)} (L_{uv})^D$	$[0, \infty)$ MBPTS	Total bandwidth capacity
$(B_{\mathrm{M}})^D = \begin{cases} \dfrac{(B_{\mathrm{T}})^D}{CON^D \cdot (N(D)^2 - N(D))}, & \text{if } CON^D > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$ MBPTS	Mean bandwidth capacity (over the links of nonzero bandwidths)
$(H_{\mathrm{B}})^D = \begin{cases} \dfrac{2}{(B_{\mathrm{T}})^D} \cdot \sqrt{\dfrac{\sum_{u=1}^{N(D)} \sum_{\substack{v=1,u\neq v,\\ (L_{uv})^D>0}}^{N(D)} ((L_{uv})^D - (B_{\mathrm{M}})^D)^2}{CON^D \cdot (N(D)^2 - N(D))}}, & \text{if } CON^D > 0 \\ 0, & \text{otherwise} \end{cases}$	[0, 1)	Bandwidth resource heterogeneity (over the interprocessor links of nonzero bandwidths)
$(LF_{uv})^D = \begin{cases} \dfrac{(L_{uv})^D}{(B_{\mathrm{T}})^D}, & \text{if } (B_{\mathrm{T}})^D > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$	Link flexibility
$(LF)^D = \begin{pmatrix} (LF_{11})^D & (LF_{12})^D & \cdots & (LF_{1,N(D)})^D \\ (LF_{21})^D & (LF_{22})^D & \cdots & (LF_{2,N(D)})^D \\ \vdots & \vdots & \ddots & \vdots \\ (LF_{N(D),1})^D & (LF_{N(D),2})^D & \cdots & (LF_{N(D),N(D)})^D \end{pmatrix}$		Link flexibility matrix
$(LF_{\mathrm{M}})^D = \dfrac{\sum_{u=1}^{N(D)} \sum_{v=1,v\neq u}^{N(D)} (LF_{uv})^D}{N(D)^2 - N(D)}$	[0, 1] MBPTS	Mean link flexibility

We can systematically add additional instance of (3.1) for capturing additional platform features. On the other hand, matrices can be ignored or removed when the corresponding resource becomes unlimited for practical issues and does not require its management (any more). Focusing on the most relevant computing resources facilitates an efficient computing resource management.

C) Platform Parameters

It may not always be possible or practical to analyze a system based on its complete modeling. Defining simple parameters that point out certain system characteristics may result useful for categorizing the system and for analyzing and understanding its behavior. On the other hand, the SDR computing resource management problem─a general mapping problem (Sections 2.3.3 and 4.1)─is too complex to be characterized by simple parameters. Those parameters may, though, be useful for recognizing similar computing resource management situations and predicting the outcomes. The parameters that we present in continuation facilitate categorizing SDR platforms and analyzing the simulation results (Chapter 6). They are derived from the above platform models.

Table 3.2 presents the computation parameters. The total processing power $(C_{\mathrm{T}})^D$ is the sum of the distributed processing capacities. An infinite processing power or resource, in general, has only theoretical significance. It symbolizes that the corresponding resource is sufficiently available, just as if its capacity were infinite. $(C_u)^D = 0$, on the other hand, indicates that all processing resources of processor $(P_u)^D$ are momentarily unavailable.

Table 3.4 Platform parameters, part IIb: communication (based on $\boldsymbol{I^D}$ and $\boldsymbol{B^D}$).

Expression	Range & Unit	Denomination
$CON^D = \begin{cases} \dfrac{\sum_{u=1}^{N(D)} \sum_{v=1, v\neq u}^{N(D)} sign\{(B_{l_{uv}})^D\}}{N(D)^2 - N(D)}, & \text{if } N(D) > 1 \\ 0, & \text{otherwise} \end{cases}$	[0, 1]	Platform's connectivity
$(B_\mathrm{T})^D = \begin{cases} \sum_{u=N(D)+1}^{N(D)\cdot N(D)} (B_u)^D, & \text{if } N(D) > 1 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$ MBPTS	Total bandwidth capacity; accumulated physical interprocessor bandwidths
$(B_\mathrm{M})^D = \begin{cases} \dfrac{(B_\mathrm{T})^D}{\sum_{u=N(D)+1}^{N(D)\cdot N(D)} sign\{(B_u)^D\}}, & \text{if } CON^D > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$ MBPTS	Mean bandwidth capacity (over the physical links of nonzero bandwidths)
$(H_\mathrm{B})^D = \begin{cases} \dfrac{2}{(B_\mathrm{T})^D} \cdot \sqrt{\dfrac{\sum_{u=N(D)+1, (B_u)^D>0}^{N(D)\cdot N(D)} ((B_u)^D - (B_\mathrm{M})^D)^2}{(B_\mathrm{T})^D / (B_\mathrm{M})^D}}, & \text{if } CON^D > 0 \\ 0, & \text{otherwise} \end{cases}$	[0, 1)	Bandwidth resource heterogeneity (over the interprocessor physical links of nonzero bandwidths)
$(LF_{uv})^D = \begin{cases} \dfrac{(B_{l_{uv}})^D}{(B_\mathrm{T})^D}, & \text{if } (B_\mathrm{T})^D > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$	Link flexibility
$\boldsymbol{LF^D} = \begin{pmatrix} (LF_{11})^D & (LF_{12})^D & \cdots & (LF_{1,N(D)})^D \\ (LF_{21})^D & (LF_{22})^D & \cdots & (LF_{2,N(D)})^D \\ \vdots & \vdots & \ddots & \vdots \\ (LF_{N(D),1})^D & (LF_{N(D),2})^D & \cdots & (LF_{N(D),N(D)})^D \end{pmatrix}$		Link flexibility matrix
$(LF_\mathrm{M})^D = \dfrac{\sum_{u=1}^{N(D)} \sum_{v=1, v\neq u}^{N(D)} (LF_{uv})^D}{N(D)^2 - N(D)}$	[0, 1] MBPTS	Mean link flexibility

We derive the *processing resource heterogeneity* $(H_\mathrm{C})^D$ from the total and mean processing capacities $(C_\mathrm{T})^D$ and $(C_\mathrm{M})^D$. It measures the variation in the processing powers through the standard deviation, where the scaling factor $2 / (C_\mathrm{T})^D$ ensures that $(H_\mathrm{C})^D \leq 1$ (Table 3.2). $(H_\mathrm{C})^D = 0$ then indicates processors of equal processing powers and $0 < (H_\mathrm{C})^D \leq 1$ a certain processing resource heterogeneity. Although completely different SDR platforms can happen to have the same processing resource heterogeneity, this parameter in conjunction with others is useful for distinguishing certain SDR platforms (Section 3.5 and Chapter 6).

The parameters describing a platform's communication features require two definitions, one assuming $\boldsymbol{L^D}$ (Table 3.3) and one assuming $\boldsymbol{I^D}$ and $\boldsymbol{B^D}$ (Table 3.4). If $\boldsymbol{L^D}$ correctly characterizes the communication network of SDR platform D, $\boldsymbol{I^D}$ and $\boldsymbol{B^D}$ can optionally be computed. Both definitions for any communication parameter are then equivalent. Otherwise, only the formulas of Table 3.4 are applicable. Some expressions use the *sign* function, which returns the sign of its argument:

$$sign\{x\} = \begin{cases} -1, & \text{if } x < 0 \\ 0, & \text{if } x = 0 \\ 1, & \text{if } x > 0 \end{cases}. \tag{3.7}$$

Table 3.5 Platform parameters, part III: communication-to-computation.

Expression	Range & Unit	Denomination
$CCR^D = \begin{cases} \frac{(B_\mathrm{T})^D}{(C_\mathrm{T})^D}, & \text{if } (B_\mathrm{T})^D > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0,\infty)$ $\frac{\mathrm{MBPTS}}{\mathrm{MOPTS}}$	Platform's Communication-to-computation ratio
$(X_{uv})^D =$ $\begin{cases} \frac{(L_{uv})^D + (L_{vu})^D}{((C_u)^D + (C_v)^D) \cdot CCR^D}, & \text{if } (L_{uv})^D + (L_{vu})^D > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$	Communication-to-computation correlation between $(P_u)^D$ and $(P_v)^D$ ($\boldsymbol{L^D}$ defined)
$\begin{cases} \frac{(B_{I_{uv}})^D + sign\lvert I_{vu} - I_{uv}\rvert \cdot (B_{I_{vu}})^D}{((C_u)^D + (C_v)^D) \cdot CCR^D}, & \text{if } (B_{I_{uv}})^D + (B_{I_{vu}})^D > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$	Communication-to-computation correlation between $(P_u)^D$ and $(P_v)^D$ ($\boldsymbol{I^D}$, $\boldsymbol{B^D}$ defined)
$\boldsymbol{X^D} = \begin{pmatrix} (X_{11})^D & (X_{12})^D & \cdots & (X_{1,N(D)})^D \\ (X_{21})^D & (X_{22})^D & \cdots & (X_{2,N(D)})^D \\ \vdots & \vdots & \ddots & \vdots \\ (X_{N(D),1})^D & (X_{N(D),2})^D & \cdots & (X_{N(D),N(D)})^D \end{pmatrix}$		Communication-to-computation correlation matrix
$(X_\mathrm{M})^D = \begin{cases} \frac{\sum_{u=1}^{N(D)-1} \sum_{v=u+1}^{N(D)} (X_{uv})^D}{(N(D)^2 - N(D))/2}, & \text{if } N(D) > 1 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$	Auxiliary variable; mean over $(X_{uv})^D \; \forall \; u, v > u$
$(X_\mathrm{STD})^D = \begin{cases} \sqrt{\frac{\sum_{u=1}^{N(D)-1} \sum_{v=u+1}^{N(D)} ((X_{uv})^D - (X_\mathrm{M})^D)^2}{(N(D)^2 - N(D))/2}}, & \text{if } N(D) > 1 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$	Platform's communication-to-computation incoherence; standard deviation of $(X_{uv})^D \; \forall \; u, v > u$

CON is our connectivity definition for SDR platforms. It is different from the *connectivity* concept in graph theory, where connectivity refers to the minimum number of edges (*edge connectivity*) or vertices (*vertex connectivity* or, simply, *connectivity*) whose removal disconnects the graph [105], [106]. CON^D relates the number of logical links that correspond or map to physical links of nonzero bandwidth capacities to the maximum number of not overlapping directional connections between $N(D)$ processors (Table 3.3 and Table 3.4). CON^D will be presented for different platform examples in Section 3.5 and applied for analyzing the mapping capabilities of SDR platforms in Chapter 6.

$(B_\mathrm{T})^D$ specifies the total bandwidth that is physically available between the processors of SDR platform D. $(B_\mathrm{M})^D$ is the platform's mean communication capacity, which is obtained averaging $(B_\mathrm{T})^D$ over the physical links of nonzero bandwidths. The bandwidth resource heterogeneity $(H_\mathrm{B})^D$ indicates whether rather similar ($(H_\mathrm{B})^D$ closer to 0) or dissimilar ($(H_\mathrm{B})^D$ closer to 1) physical bandwidth capacities characterize platform D. It is obtained as the scaled standard deviation of the physical link bandwidths. This parameter facilitates distinguishing platforms with equivalent communication networks except for the distribution of bandwidth resources (Chapter 6).

Table 3.3 and Table 3.4 also introduce the *link flexibility* $(LF_{uv})^D$ ($u, v \in 1, 2, \ldots, N(D)$). It divides the maximum directional communication capacity of the logical link between a processor pair by the platform's total bandwidth capacity $(B_\mathrm{T})^D$. The link flexibility of a processor-internal link informs about the processor's internal data flow capacity with respect to $(B_\mathrm{T})^D$. The *link flexibility matrix* $\boldsymbol{LF^D}$ captures the flexibilities of all logical links of SDR platform D, whereas the *mean link flexibility* $(LF_\mathrm{M})^D$ abstracts the platform's link flexibility. These parameters facilitate distinguishing and qualifying different types of interprocessor communication lines and networks (Section 3.5 and Chapter 6).

Heterogeneous computing literature defines the communication-to-computation ratio (CCR) as the application's average communication cost divided by the average computation cost on a given system [42]. This parameter is used for (randomly) generating a number of multiprocessor scheduling problems, conducting simulations, and comparing scheduling algorithms [44], [46], [55], [61], [79], [81]. Our modular system modeling, however, suggests defining the CCR separately for platforms and applications. We can then use both parameters together for simulating different problems and analyzing the results.

CCR^D is the platform's CCR. It relates platform D's total communication resources to its total computing resources (Table 3.5).The communication-to-computation correlation (CCC) matrix $\boldsymbol{X^D}$ extends our CCR concept. It captures the *local* communication-to-computation ratios: $(X_{uv})^D$ (u, $v \in 1, 2, \ldots, N(D)$) relates the bidirectional communication link capacity between the processors pair $(P_u)^D$ and $(P_v)^D$ to the scaled sum of their processing powers, where the scaling eliminates the units (Table 3.5). $\boldsymbol{X^D}$ is symmetric to the main diagonal. The expression $sign|x|$, where $|x|$ stands for the absolute value of x, is either 0 or 1 (3.7). It avoids considering HD links twice.

$(X_M)^D$ is an auxiliary variable that is used for computing the *platform's communication-to-computation incoherence* $(X_{STD})^D$. $(X_{STD})^D$ quantifies the platform's resource distribution: the higher $(X_{STD})^D$ the less coherent the distribution of the bandwidth capacities with respect to the processing resources. A low $(X_{STD})^D$ indicates that the total bandwidth capacity is distributed in due proportion to the processing powers. The suitability of these parameters will also be demonstrated in Chapter 6.

3.3.2 Application Modeling

A) Application Templates

The $x(t'')^d$ times $y(t'')^d$ matrix $(\boldsymbol{R}_A^{t''})^{\boldsymbol{d}}$,

$$(\boldsymbol{R}_A^{t''})^{\boldsymbol{d}} = \begin{pmatrix} (R_A^{t''}|_{11})^d & (R_A^{t''}|_{12})^d & \cdots & (R_A^{t''}|_{1,y(t'')^d})^d \\ (R_A^{t''}|_{21})^d & (R_A^{t''}|_{22})^d & \cdots & (R_A^{t''}|_{1,y(t'')^d})^d \\ \vdots & \vdots & \ddots & \vdots \\ (R_A^{t''}|_{x(t'')^d,1})^d & (R_A^{t''}|_{x(t'')^d,2})^d & \cdots & (R_A^{t''}|_{x(t'')^d,y(t'')^d})^d \end{pmatrix}, \tag{3.8}$$

models the application environment, where $t'' \in 1, 2, \ldots, T''$ is the application feature index and d the SDR application index (Table 3.1). It is the general template for modeling SDR applications' computing characteristics. Equivalently to the computing resource template $(\boldsymbol{R^t})^{\boldsymbol{D}}$ (3.2), we define the subtemplate

$$(\boldsymbol{r^t})^{\boldsymbol{d}} = \begin{pmatrix} (r_{11}^t)^d & (r_{12}^t)^d & \cdots & (r_{1,y(t)^d}^t)^d \\ (r_{21}^t)^d & (r_{22}^t)^d & \cdots & (r_{2,y(t)^d}^t)^d \\ \vdots & \vdots & \ddots & \vdots \\ (r_{x(t)^d,1}^t)^d & (r_{x(t)^d,2}^t)^d & \cdots & (r_{x(t)^d,y(t)^d}^t)^d \end{pmatrix} \tag{3.9}$$

for capturing the computing resource requirements of SDR applications.

B) Application Models

A concise characterization of SDR applications requires several instances of (3.8). We consider processing and data flow requirements as the principal computing requirements of SDR applications and introduce the *function model*, several *data flow models*, and the *stage model*.

SDR application d consists of $M(d)$ SDR functions: $(f_1)^d$, $(f_2)^d$, …, $(f_{M(d)})^d$ (Table 3.1). The function model

$$(\boldsymbol{R}_A^{\boldsymbol{1}})^{\boldsymbol{d}} = (\boldsymbol{r^1})^{\boldsymbol{d}} = \boldsymbol{c^d} = ((c_1)^d, (c_2)^d, \ldots, (c_{M(d)})^d) \text{ MOPTS} \tag{3.10}$$

provides the processing requirements of $(f_1)^d$, $(f_2)^d$, …, $(f_{M(d)})^d$.

The data flow model

$$(\boldsymbol{R}_{\mathrm{A}}{}^{2})^{d} = (\boldsymbol{r}^{2})^{d} = \boldsymbol{l}^{d} = \begin{pmatrix} (l_{11})^{d} & (l_{12})^{d} & \cdots & (l_{1,M(d)})^{d} \\ (l_{21})^{d} & (l_{22})^{d} & \cdots & (l_{2,M(d)})^{d} \\ \vdots & \vdots & \ddots & \vdots \\ (l_{M(d),1})^{d} & (l_{M(d),2})^{d} & \cdots & (l_{M(d),M(d)})^{d} \end{pmatrix} \text{MBPTS} \tag{3.11}$$

corresponds to the communication model $\boldsymbol{L}^{\boldsymbol{D}}$ (3.4). Element $(l_{uv})^{d}$ represents the minimum necessary bandwidth for sending a certain data amount from $(f_u)^{d}$ to $(f_v)^{d}$ ($u, v \in 1, 2, \ldots, M(d)$; $(l_{uv})^{d} \in \mathbb{R}^{+}$). It implicitly indicates the precedence constraints between SDR functions.

We assume that SDR processing chains represent DAGs. General-purpose as well as digital signal processing chains are typically modeled as DAGs [40], [49], [53], [86]. Cycles and loops, which are common in signal processing, are then either unrolled [52], embedded (inner loops – SDR function level), or handled through a proper time management based on the principles of Section 3.2.2 (outer loops – SDR application level). DAGs can be logically numbered: If $(f_u)^{d}$ sends data to $(f_v)^{d}$, then $u < v$ [104]. The dataflow model then becomes a strictly upper diagonal matrix, where $(l_{uv})^{d} = 0 \; \forall \; u \geq v$.

Although the above data flow model is appropriate for DAGs, we can define a pair of models that corresponds to the communication models (3.5) and (3.6):

$$(\boldsymbol{R}_{\mathrm{A}}{}^{3})^{d} = \boldsymbol{i}^{d} = \begin{pmatrix} (i_{11})^{d} & (i_{12})^{d} & \cdots & (i_{1,M(d)})^{d} \\ (i_{21})^{d} & (i_{22})^{d} & \cdots & (i_{2,M(d)})^{d} \\ \vdots & \vdots & \ddots & \vdots \\ (i_{M(d),1})^{d} & (i_{M(d),2})^{d} & \cdots & (i_{M(d),M(d)})^{d} \end{pmatrix} \tag{3.12}$$

and

$$(\boldsymbol{R}_{\mathrm{A}}{}^{4})^{d} = (\boldsymbol{r}^{3})^{d} = \boldsymbol{b}^{d} = ((b_1)^{d}, (b_2)^{d}, \ldots, (b_{M(d)\cdot M(d)})^{d}) \text{ MBPTS.} \tag{3.13}$$

These equations model the data flow requirements as follows: The minimum bandwidth that is necessary for the real-time data transfer from $(f_u)^{d}$ to $(f_v)^{d}$ is $(b_n)^{d}$, where $n = (i_{uv})^{d} \in 1, 2, \ldots, M(d) \cdot M(d)$ and $(b_n)^{d} \in \mathbb{R}^{+}$. Then, $(b_n)^{d} = 0$ indicates that there is no data flow link from $(f_u)^{d}$ to $(f_v)^{d}$. We can organize $\boldsymbol{i}^{d}$ so that $(b_1)^{d} \geq (b_2)^{d} \geq \ldots \geq (b_{M(d)\cdot M(d)})^{d}$, where the last $M(d) \cdot (M(d)+1)/2$ elements are 0 for $M(d)$-node DAGs.

The above distinction between precedence constraints and bandwidth demands facilitates distinguishing between dependent and independent data flows: If process f_1, for example, sends the same data to f_3 and f_4, i_{13} and i_{14} would point to the same entry in $\boldsymbol{b}^{d}$ ($i_{13} = i_{14}$ – dependent data flows), whereas if f_1 sends two different data chunks, one to f_3 and another to f_4, i_{13} and i_{14} should point to the different entries in $\boldsymbol{b}^{d}$ ($i_{13} \neq i_{14}$ – independent data flows). Despite this modeling capability, our computing resource management proposal does not distinguish between dependent and independent data flows, but, rather, assumes that all data flows are independent.

As indicated in Section 3.2.1, the above models depend on the assumed processor type, the optimization level, and so forth. Therefore, $\boldsymbol{c}^{d}$, $\boldsymbol{l}^{d}$, $\boldsymbol{i}^{d}$, and $\boldsymbol{b}^{d}$ need to be available for each one of the platform's processors. Without loss of generality and to improve its readability, this dissertation treats each of these models as if it were unique for a given SDR platform rather than processor-specific.

The direct predecessors of $(f_n)^{d}$ are all those SDR functions that correspond to nonzero entries in the n^{th} column of $\boldsymbol{l}^{d}$ ($n \in 1, 2, \ldots, M(d)$). Hence, columns of all 0s in (3.11) indicate *source functions*. Correspondingly, the nonzero entries in the n^{th} row of $\boldsymbol{l}^{d}$ determine the direct successors of $(f_n)^{d}$. If the n^{th} row of $\boldsymbol{l}^{d}$ contains only 0s, $(f_n)^{d}$ has no successor and is called a *sink function* [104]. These concepts are also valid when using the data flow models $\boldsymbol{i}^{d}$ and $\boldsymbol{b}^{d}$ and lead to the stage model

$$(\boldsymbol{R}_{\mathrm{A}}{}^{5})^{d} = \boldsymbol{s}^{d} = ((s_1)^{d}, (s_2)^{d}, \ldots, (s_{M(d)})^{d}). \tag{3.14}$$

This model specifies the pipelining stages of SDR application d, where the pipelining stage of SDR function $(f_u)^{d}$ is $(s_u)^{d}$ ($(s_u)^{d} \in \mathbb{N}$).

An SDR function $(f_u)^{d}$ is assigned to stage $(s_u)^{d} = n$ if it receives data from no other SDR function than from those in stages n–1, n–2, ..., 1, and if at least one of its direct predecessors is in stage n–1. All source functions are first assigned to pipelining stage 1, their direct successors to stage 2, and so forth. The stage model is useful for synchronizing the data processing through the execution pipeline.

Our pipelining stages stem from the *levels* concept in graph theory. The level of a node in a DAG reflects the longest path of which it is the last point [104]. Pipelining stage n then corresponds to level n–1. An SDR application has a unique stage model, whereas it can have many logical numberings.

C) Application Parameters

Several parameters can be derived from the above application models. Here we present those that correspond to the platform parameters of Section 3.3.1C) but also others that are specific to SDR applications. Most of these parameters will result practical for generating random DAGs and for analyzing the simulation results of Chapter 6.

Table 3.6 summarizes the computation parameters of SDR applications. They correspond to the computation parameters of SDR platforms due to Table 3.2: An SDR application's total processing requirement $(c_\mathrm{T})^d$ is the sum of the processing requirements of its SDR functions. From the total and mean processing requirements we can compute the *processing demand heterogeneity* $(h_\mathrm{C})^d$. It indicates whether the processing demands $(c_1)^d$ through $(c_{M(d)})^d$ are rather similar or dissimilar. In particular, $(h_\mathrm{C})^d = 0$, if $(c_1)^d = (c_2)^d = \ldots = (c_{M(d)})^d$ and $0 < (h_\mathrm{C})^d \leq 1$, otherwise.

Table 3.7 and Table 3.8 present a few parameters derived from the data flow models (3.11), (3.12), and (3.13). These parameters assume DAG representations of SDR applications and treat all data flows as if they were independent. Since $\boldsymbol{l}^d$ can correctly characterize any DAG, Table 3.8 is optional and any definition based on $\boldsymbol{i}^d$ and $\boldsymbol{b}^d$ is equivalent to the corresponding one based on $\boldsymbol{l}^d$.

Table 3.6 Application parameters, part I: computation.

Expression	Range & Unit	Denomination
$(c_\mathrm{T})^d = (c_1)^d + (c_2)^d + \ldots + (c_{M(d)})^d$	$[0, \infty)$ MOPTS	Total processing demand
$(c_\mathrm{M})^d = \dfrac{(c_\mathrm{T})^d}{M(d)}$	$[0, \infty)$ MOPTS	Mean processing demand
$(h_\mathrm{C})^d = \begin{cases} \dfrac{2}{(c_\mathrm{T})^d} \cdot \sqrt{\dfrac{1}{M(d)} \cdot \sum_{u=1}^{M(d)} ((c_u)^d - (c_\mathrm{M})^d)^2}, & \text{if } (c_T)^d > 0 \\ 0, & \text{otherwise} \end{cases}$	[0, 1]	Processing demand heterogeneity

Table 3.7 Application parameters, part IIa: communication (based on $\boldsymbol{l}^d$).

Expression	Range & Unit	Denomination
$con^d = \begin{cases} \dfrac{\sum_{u=1}^{M(d)-1} \sum_{v=u+1}^{M(d)} sign\{(l_{uv})^d\}}{(M(d)^2 - M(d))/2}, & \text{if } M(d) > 1 \\ 0, & \text{otherwise} \end{cases}$	[0, 1]	Application's connectivity
$(b_\mathrm{T})^d = \begin{cases} \sum_{u=1}^{M(d)-1} \sum_{v=u+1}^{M(d)} (l_{uv})^d, & \text{if } M(d) > 1 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$ MBPTS	Total bandwidth demand
$(b_\mathrm{M})^d = \begin{cases} \dfrac{(b_\mathrm{T})^d}{con^d \cdot (M(d)^2 - M(d))/2}, & \text{if } con^d > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$ MBPTS	Mean bandwidth demand
$(h_\mathrm{B})^d = \begin{cases} \dfrac{2}{(b_\mathrm{T})^d} \cdot \sqrt{\dfrac{\sum_{u=1}^{M(d)-1} \sum_{\substack{v=u+1, \\ (l_{uv})^d > 0}}^{M(d)} ((l_{uv})^d - (b_\mathrm{M})^d)^2}{con^d \cdot (M(d)^2 - M(d))/2}}, & \text{if } con^d > 0 \\ 0, & \text{otherwise} \end{cases}$	[0, 1)	Bandwidth demand heterogeneity (Heterogeneity of the independent and nonzero bandwidth demands)

Table 3.8 Application parameters, part IIb: communication (based on $\boldsymbol{i}^d$ and $\boldsymbol{b}^d$).

Expression	Range & Unit	Denomination
$con^d = \begin{cases} \dfrac{\sum_{u=1}^{M(d)-1}\sum_{v=u+1}^{M(d)} sign\{(b_{i_{uv}})^d\}}{(M(d)^2 - M(d))/2}, & \text{if } M(d) > 1 \\ 0, & \text{otherwise} \end{cases}$	[0, 1]	Application's connectivity
$(b_\mathrm{T})^d = \begin{cases} \sum_{u=1}^{M(d)-1}\sum_{v=u+1}^{M(d)} (b_{i_{uv}})^d, & \text{if } M(d) > 1 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$ MBPTS	Total bandwidth demand
$(b_\mathrm{M})^d = \begin{cases} \dfrac{(b_\mathrm{T})^d}{con^d \cdot (M(d)^2 - M(d))/2}, & \text{if } con^d > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0, \infty)$ MBPTS	Mean bandwidth demand
$(h_\mathrm{B})^d = \begin{cases} \dfrac{2}{(b_\mathrm{T})^d} \cdot \sqrt{\dfrac{\sum_{u=1}^{M(d)-1}\sum_{\substack{v=u+1, \\ (b_{i_{uv}})^d>0}}^{M(d)} ((b_{i_{uv}})^d - (b_\mathrm{M})^d)^2}{con^d \cdot (M(d)^2 - M(d))/2}}, & \text{if } con^d > 0 \\ 0, & \text{otherwise} \end{cases}$	[0, 1)	Bandwidth demand heterogeneity (Heterogeneity over the non-zero bandwidth demands)

Table 3.9 Application parameters, part IIc: pipelining.

Expression	Range & Unit	Denomination
$(n_\mathrm{TS})^d = \max_{u=1,2,..,M(d)} \left\{ (s_u)^d \right\}$	$[1, M(d)]$	Number of pipelining stages
$g^d = \dfrac{(n_\mathrm{TS})^d}{M(d)}$	$[M(d)^{-1}, 1]$	Application length

The application's connectivity con^d relates the number of dedicated directional connections of nonzero bandwidth demands to the maximum number of logical links of $M(d)$-node DAGs. It captures the relative number of data flow connections. The absolute number of data flow links with nonzero bandwidth requirements is then $con^d \cdot (M(d)^2 - M(d))/2$. The total bandwidth demand $(b_\mathrm{T})^d$ is the sum of all data flow requirements. The mean bandwidth demand $(b_\mathrm{M})^d$ then captures the application's average data flow demand. With these parameters we can compute the *bandwidth demand heterogeneity* $(h_\mathrm{B})^d$. It quantifies the data flow dissimilarities of the existing data flow links normalized to [0, 1) (Table 3.7 and Table 3.8).

We could define a set of application parameters that correspond to the link flexibility $(LF_{uv})^D$, the link flexibility matrix $\boldsymbol{LF}^{\boldsymbol{D}}$, and the mean link flexibility $(LF_\mathrm{M})^D$ of SDR platforms (Table 3.3 and Table 3.4). Since assuming DAGs, where any data flow link is directional and connects exactly two SDR functions, the link flexibility or, rather, *data flow concentration* of an SDR application's data flow link would inform about the link's bandwidth demand with respect to the application's total bandwidth requirement $(b_\mathrm{T})^d$. The corresponding *data flow concentration matrix* would then be a scaled version of $\boldsymbol{l}^{\boldsymbol{d}}$.

Table 3.9 defines two useful pipelining parameters. The first parameter specifies the number of pipelining stages of SDR application d as a function of the stage vector $\boldsymbol{s}^{\boldsymbol{d}}$. Except for DAGs with a single end node, $(n_\mathrm{TS})^d \neq (s_{M(d)})^d$, in general. The *length* of application d relates $(n_\mathrm{TS})^d$ to $M(d)$. A value of g^d close to $M(d)^{-1}$ indicates a high degree of parallelism, or a *short* SDR application, whereas a value closer to 1 implies more sequential dependencies and, hence, a *longer* application. The task graph representation of an SDR application visually reflects this (Section 3.5).

Table 3.10 defines the communication-to-computation parameters of SDR applications. With the application's CCR ccr^d we can compute the application's CCC matrix $\boldsymbol{x}^{\boldsymbol{d}}$, which contains the local CCRs. It is symmetric to the main diagonal. Because of the acyclic data flow dependencies, the diagonal elements are 0 and at most one summand in the numerator of $(x_{uv})^d$ can be nonzero. The remaining two parameters—$(x_\mathrm{M})^d$ and $(x_\mathrm{STD})^d$—and their definitions correspond to $(X_\mathrm{M})^D$ and $(X_\mathrm{STD})^D$ of Table 3.5.

Table 3.10 Application parameters, part III: communication-to-computation.

Expression	Range & Unit	Denomination
$ccr^{\,d} = \begin{cases} \dfrac{(b_{\mathrm{T}})^d}{(c_{\mathrm{T}})^d}, & \text{if } (b_{\mathrm{T}})^d > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0,\infty)$ $\frac{\text{MBPTS}}{\text{MOPTS}}$	Application's communication-to-computation ratio
$(x_{uv})^d =$ $\begin{cases} \dfrac{(l_{uv})^d + (l_{vu})^d}{((c_u)^d + (c_v)^d)\cdot ccr^{\,d}}, & \text{if } (l_{uv})^d + (l_{vu})^d > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0,\infty)$	Communication-to-computation correlation between $(f_u)^{\mathrm{d}}$ and $(f_v)^{\mathrm{d}}$ ($\boldsymbol{l}^{\boldsymbol{d}}$ defined)
$\begin{cases} \dfrac{(b_{i_{uv}})^d + (b_{i_{vu}})^d}{((c_u)^d + (c_v)^d)\cdot ccr^{\,d}}, & \text{if } (b_{i_{uv}})^d + (b_{i_{vu}})^d > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0,\infty)$	Communication-to-computation correlation between $(f_u)^{\mathrm{d}}$ and $(f_v)^{\mathrm{d}}$ ($\boldsymbol{i}^{\boldsymbol{d}}$, $\boldsymbol{b}^{\boldsymbol{d}}$ defined)
$\boldsymbol{x}^{\boldsymbol{d}} = \begin{pmatrix} (x_{11})^d & (x_{12})^d & \cdots & (x_{1,M(d)})^d \\ (x_{21})^d & (x_{22})^d & \cdots & (x_{2,M(d)})^d \\ \vdots & \vdots & \ddots & \vdots \\ (x_{M(d),1})^d & (x_{M(d),2})^d & \cdots & (x_{M(d),M(d)})^d \end{pmatrix}$		Application's communication-to-computation correlation matrix
$(x_{\mathrm{M}})^d = \begin{cases} \dfrac{\sum_{u=1}^{M(d)-1}\sum_{v=u+1}^{M(d)}(x_{uv})^d}{(M(d)^2 - M(d))/2}, & \text{if } M(d) > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0,\infty)$	Auxiliary variable; mean over $(x_{uv})^d\ \forall\ u, v > u$
$(x_{\mathrm{STD}})^d = \begin{cases} \sqrt{\dfrac{\sum_{u=1}^{M(d)-1}\sum_{v=u+1}^{M(d)}((x_{uv})^d - (x_{\mathrm{M}})^d)^2}{(M(d)^2 - M(d))/2}}, & \text{if } M(d) > 0 \\ 0, & \text{otherwise} \end{cases}$	$[0,\infty)$	Application's communication-to-computation incoherence; standard deviation of $(x_{uv})^d\ \forall\ u, v > u$

3.4 Meeting the SDR Computing Constraints

The computing resource management facilities of Section 3.2 and the system modeling of Section 3.3 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 % 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 [40], [41], [42], [43], [44], 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. The SDR framework finally needs to manage the synchronized execution on all processors and provide pipelining and buffering mechanisms, among others, for the proper and timely data delivery. The P-HAL-OE provides all these mechanisms (Section 2.2.3).

The usually complex scheduling process can, on the basis of a feasible mapping and under the above assumptions, be simplified to $N(D)$ independent local scheduling tasks. A processor's local scheduler is then 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. 3.2). This ensures that the input data of any SDR application's module or set of modules are processed according to the arrival rate so that no data are accumulated anywhere in the processing chain, meeting the *minimum bit rate requirement* (Section 2.3.2).

The application-specific time slot duration $(t_{TS})^d$ times the number of pipelining stages $(n_{TS})^d$ is the pipelining latency in case of a feasible schedule on each processor. We specify $(t_{TS})^d$ as

$$(t_{TS})^d = \frac{(L_{MAX})^d}{(n_{TS})^d} \text{ SPTS} \tag{3.15}$$

to meet the application's tolerable latency $(L_{MAX})^d$, 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 end user.

When more than one SDR application needs to be executed on SDR platform D, the common time slot duration t_{TS} must satisfy all latency demands. Hence, the minimum $(t_{TS})^d$ of all applications that need to be concurrently executed on SDR platform D specifies t_{TS}. We assume that t_{TS} is large enough for the (efficient) execution of any SDR function in the processing chain or chains.

On another management level, access to any shared resource requires its temporal management or scheduling. Each shared link, for example, requires a *bus scheduler*. Assuming the availability of data buffers, a bus scheduler can use a simple policy to ensure timely data transfers: Transfer data from the output data buffer of the origin processor to the input buffer of the destination processor as soon as the bus becomes available, gaining access to the different data transfers in a round-robin fashion, for instance.

We summarize that $N(D)$ processor-local schedulers need to schedule processes, data transfers, and possibly other resource allocations for guaranteeing that real-time constraints will eventually be met. On the basis of a feasible mapping, finding such schedules is possible and simple if we assume that processing chains can be pipelined, that data processing and data transfers can overlap, and that partial results can be immediately forwarded to the next processing block. Our SDR computing resource management framework, though, facilitates elaborating more sophisticated scheduling approaches or different execution patterns for reducing the pipelining latency.

3.5 Modeling Examples

Fig. 3.3 shows four processing platforms. Three (pseudo-)homogeneous processors and a homogeneous communication network characterize SDR platform I (Fig. 3.3a). Platform II differs from I in that its interprocessor bandwidths are heterogeneous (Fig. 3.3b). Platforms III and IV are equivalent to platforms I and II except for the heterogeneous processors (Fig. 3.3c and d).

The three processors of SDR platform I, II, III, or IV may stand for three individual processing elements 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 [25].

Fig. 3.4 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) [107]. The computing requirements are estimates from [107], [108], [110] and available implementations using the *TMS320C6416* DSP and the *Code Composer Studio* from Texas Instruments [111]. The processing requirements have been obtained from the number of MACs times the sampling rate f_S. The product between f_S and the bit precision of $2 \cdot 16$ bits for the real and the imaginary components correspondingly specifies a data flow demands.

We compute the stage model (3.14) of the processing chain of Fig. 3.4 and obtain the number of pipelining stages as $(n_{TS})^{UMTS} = 17$. We consider $L_{MAX} = 10$ ms (the UMTS radio link uses 10 ms long frames to synchronize the data transmission and reception) and obtain a time slot duration of $(t_{TS})^{UMTS} = 0.01 / 17 = 0.588 \cdot 10^{-3}$ SPTS due to (3.15). Fig. 3.5 presents the platform models of SDR platform IV (Fig. 3.3d) and Fig. 3.7 the application models of the UMTS downlink receiver of Fig. 3.4. Index *UMTS* here refers to the specific UMTS downlink receiver implementation.

Since SDR platform IV contains only dedicated directional communication lines (Fig. 3.5a), $\boldsymbol{L}^{IV}$ is suitable for modeling the communication network and contains the same information as $\boldsymbol{I}^{IV}$ and $\boldsymbol{B}^{IV}$ together (Fig. 3.5b). The corresponding platform parameters are given in Table 3.11. The mean link flexibility of such an interconnection network is 1/6, because the total bandwidth is distributed among six physical links. It then follows that $(LF_M)^I = (LF_M)^{II} = (LF_M)^{III} = (LF_M)^{IV} = 1/6$.

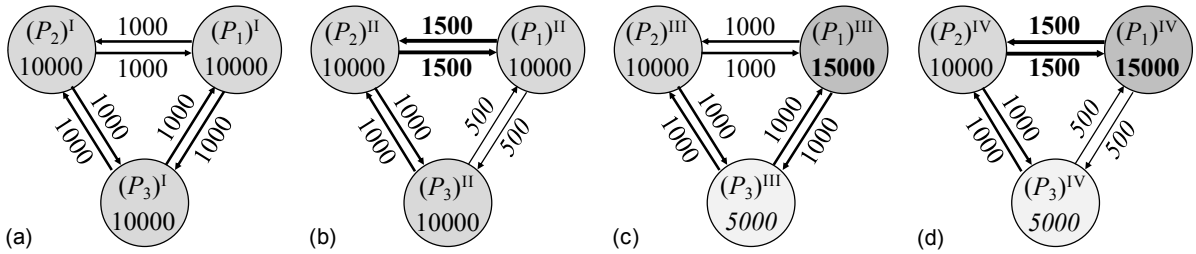


Fig. 3.3. Processing and bandwidth resources in MOPS and Mbps of SDR platforms I-IV (a)-(d).

The four SDR platforms I, II, III, and IV are distinguishable through the processing and bandwidth resource heterogeneities (Fig. 3.6). Besides, the CCR matrices and their parameters are platform specific; that is, $\boldsymbol{X}^{\mathrm{I}} \neq \boldsymbol{X}^{\mathrm{II}} \neq \boldsymbol{X}^{\mathrm{III}} \neq \boldsymbol{X}^{\mathrm{IV}}$. Chapters 5 and 6 will analyze the implications of these platform differences.

The UMTS task graph and its mathematical modeling are shown in Fig. 3.7. The data flow model $\boldsymbol{l}^{\mathrm{UMTS}}$ indicates the data flows between SDR functions and the corresponding bandwidth requirements: $(l_{uv})^{\mathrm{UMTS}}$ is the minimum bandwidth demand for the data flow from $(f_u)^{\mathrm{UMTS}}$ to $(f_v)^{\mathrm{UMTS}}$. A zero bandwidth requirement then symbolizes a not existing data flow link.

As opposed to $\boldsymbol{l}^{\mathrm{UMTS}}$, $\boldsymbol{i}^{\mathrm{UMTS}}$ and $\boldsymbol{b}^{\mathrm{UMTS}}$ distinguish between dependent and independent data flows. The two data flows between SDR functions f_1, f_2, and f_3, for example, are independent and are labeled as *link 1* and *link 2*: $(i_{12})^{\mathrm{UMTS}} = 1$ and $(i_{13})^{\mathrm{UMTS}} = 2$. SDR function f_4, on the other hand, sends the same data to f_6, f_7, f_8, f_9, and f_{11}. The five corresponding entries in $\boldsymbol{i}^{\mathrm{UMTS}}$ then point to the same element in $\boldsymbol{b}^{\mathrm{UMTS}}$ (Fig. 3.7b).

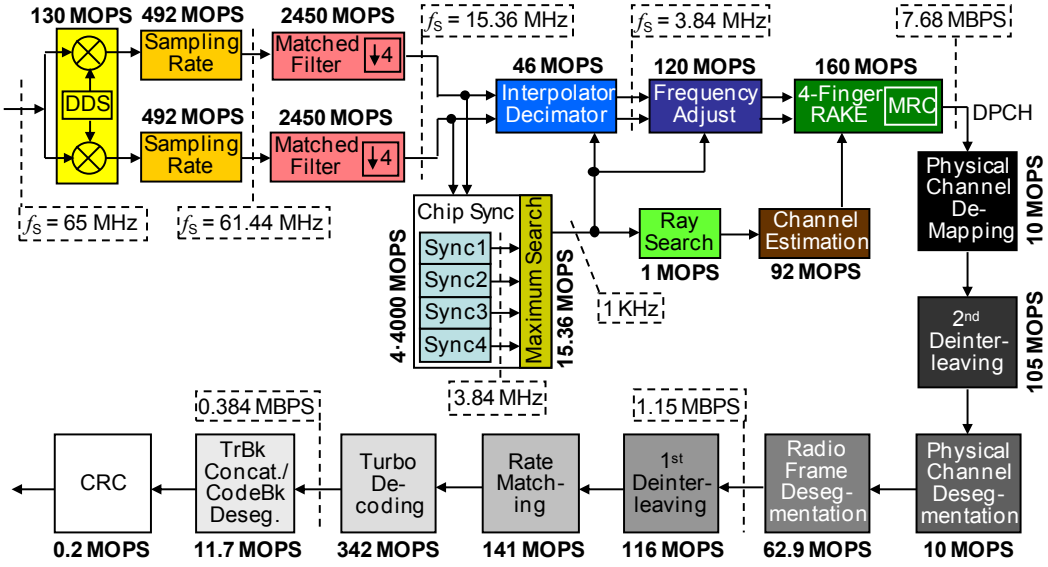
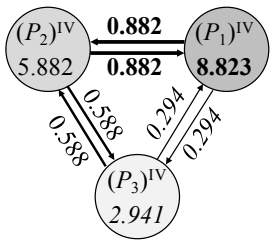


Fig. 3.4. Functional diagram and requirements of a UMTS downlink receiver.



(a)

Device model

$\boldsymbol{C}^{\mathrm{IV}} = (8.823, 5.882, 2.941)$ MOPTS

Communication models

$$\boldsymbol{L}^{\mathrm{IV}} = \begin{pmatrix} \infty & 0.882 & 0.294 \\ 0.882 & \infty & 0.588 \\ 0.294 & 0.588 & \infty \end{pmatrix} \text{MBPTS}$$

$$\boldsymbol{I}^{\mathrm{IV}} = \begin{pmatrix} 1 & 4 & 8 \\ 5 & 2 & 6 \\ 9 & 7 & 3 \end{pmatrix}$$

$\boldsymbol{B}^{\mathrm{IV}} = (\infty, \infty, \infty, 0.882, 0.882, 0.588, 0.588, 0.294, 0.294)$ MBPTS

(b)

Fig. 3.5. Platform models of SDR platform IV: graphical (a) and mathematical (b) representations.

Table 3.11 Modeling parameters of SDR platform IV (Fig. 3.5).

Computation	Communication	Communication-to-computation
$N(\mathrm{IV}) = 3$ $(C_\mathrm{T})^\mathrm{IV} = 17.647$ MOPTS $(C_\mathrm{M})^\mathrm{IV} = 5.882$ MOPTS $(H_\mathrm{C})^\mathrm{IV} = 0.272$	$CON^\mathrm{IV} = 1$ $(B_\mathrm{T})^\mathrm{IV} = 3.529$ MBPTS $(B_\mathrm{M})^\mathrm{IV} = 0.588$ MBPTS $(H_\mathrm{B})^\mathrm{IV} = 0.136$ $\boldsymbol{LF}^\mathbf{IV} = \begin{pmatrix} \infty & 1/4 & 1/12 \\ 1/4 & \infty & 1/6 \\ 1/12 & 1/6 & \infty \end{pmatrix}$ $(LF_\mathrm{M})^\mathrm{IV} = 1/6$	$CCR^\mathrm{IV} = 0.2$ MBPTS / MOPTS $\boldsymbol{X}^\mathbf{IV} = \begin{pmatrix} \infty & 0.6 & 0.25 \\ 0.6 & \infty & 0.666 \\ 0.25 & 0.666 & \infty \end{pmatrix}$ $(X_\mathrm{M})^\mathrm{IV} = 0.506$ $(X_\mathrm{STD})^\mathrm{IV} = 0.183$

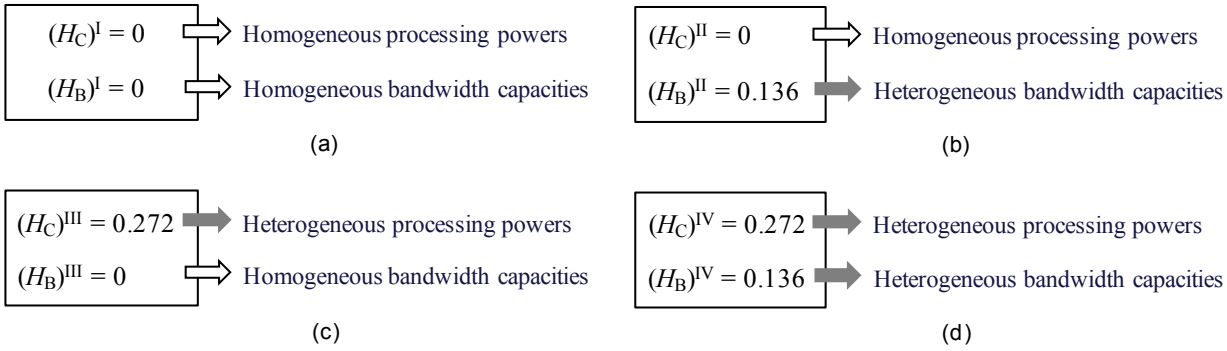


Fig. 3.6. Resource heterogeneity parameters of SDR platforms I-IV (a)-(d).

Out of the 34 existing data flows, 24 are independent. Then, only 24 entries in $\boldsymbol{b}^\mathbf{UMTS}$ are nonzero; these are the first 24 entries, since $\boldsymbol{b}^\mathbf{UMTS}$ is ordered by decreasing bandwidth demands. The elements of $\boldsymbol{i}^\mathbf{UMTS}$ that correspond to not existing data flow links could point to any element in $\boldsymbol{b}^\mathbf{UMTS}$ between $(b_{25})^\mathrm{UMTS}$ and $(b_{576})^\mathrm{UMTS}$. (The additional information of $\boldsymbol{i}^\mathbf{UMTS}$ and $\boldsymbol{b}^\mathbf{UMTS}$ with respect to $\boldsymbol{l}^\mathbf{UMTS}$ is not processed in this dissertation.)

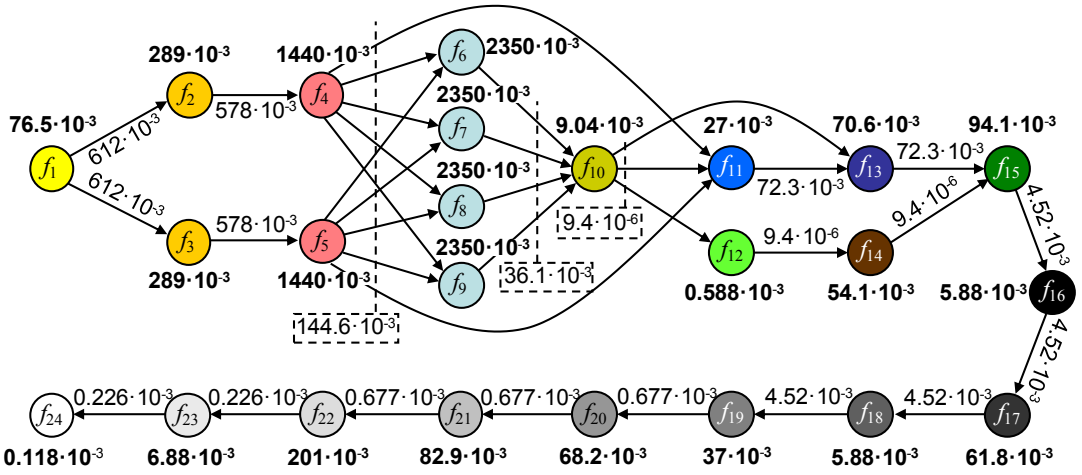
Table 3.12 and Fig. 3.8 present the application parameters of the UMTS receiver models. They complete the SDR application example here, whereas their utility will be discussed in later chapters. Note, however, that $g^\mathrm{UMTS} \approx 0.7$ indicates a rather sequential than parallel processing chain (Section 3.3.2C)); the task graph of Fig. 3.7a visually reflects this.

Fig. 3.9 illustrates another SDR platform modeling example. Here all processors share two unidirectional buses (Fig. 3.9a). Two unidirectional buses equivalently connect each cluster of four processors with the rest of the processing array of picoChips's picoArray *PC101* [25].

Fig. 3.9b and c show the communication models and parameters of Fig. 3.9a. Since links are shared here, the communication models $\boldsymbol{I}^\mathbf{V}$ and $\boldsymbol{B}^\mathbf{V}$ need to be used instead of $\boldsymbol{L}^\mathbf{V}$. A set of four logical interprocessor links map to one unidirectional physical link. Since both physical links offer the same bandwidth, the bandwidth resource heterogeneity $(H_\mathrm{B})^\mathrm{V}$ is 0. The link flexibility of each logical link is 0.5, because the maximal directional bandwidth between each pair of processor is $0.5 \cdot (B_\mathrm{T})^\mathrm{V}$. Hence, the mean link flexibility of such an SDR platform is 0.5.

SDR platform VI connects each pair of processors through bidirectional HD links (Fig. 3.10a). Fig. 3.10b and c capture the communication models and parameters of this platform. The pair of logical links between any two processors maps to a single bidirectional physical link. The mean link flexibility of such a processor platform is 1/3, irrespective of the actual bandwidths B_x, B_y, and B_z. Even if the capacity of one or two HD links were zero, $\boldsymbol{LF}^\mathbf{VI}$ and $\boldsymbol{CON}^\mathbf{VI}$ would change but not $(LF_\mathrm{M})^\mathrm{VI}$.

Fig. 3.11a finally shows a communication network that cannot be characterized by the communication models $\boldsymbol{I}^\boldsymbol{D}$ and $\boldsymbol{B}^\boldsymbol{D}$. One unidirectional and one bidirectional link connects the two processors $(P_1)^\mathrm{VII}$ and $(P_2)^\mathrm{VII}$ (Fig. 3.11a). The total bandwidth capacity from $(P_2)^\mathrm{VII}$ to $(P_1)^\mathrm{VII}$ is $B_{bus} + B_{link}$, part of which is shared. $(I_{12})^\mathrm{VII}$ and $(I_{21})^\mathrm{VII}$ then both need to point to the bidirectional bus bandwidth B_{bus}, while $(I_{21})^\mathrm{VII}$ also needs to point to the unidirectional link bandwidth B_{link}. $\boldsymbol{I}^\boldsymbol{D}$ would need a third dimension for modeling such a communication network.



(a)

Function model

$\boldsymbol{c}^{\mathrm{UMTS}} = (76.5, 289, 289, 1440, 1440, 2350, 2350, 2350, 2350, 9.04, 27, 0.588,$
$70.6, 54.1, 94.1, 5.88, 61.8, 5.88, 37, 68.2, 82.9, 201, 6.88, 0.118) \cdot 10^{-3}$ MOPTS

Dataflow models

$$\boldsymbol{l}^{\mathrm{UMTS}} = \begin{pmatrix}
0 & 612 & 612 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 578 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 578 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 144.6 & 144.6 & 144.6 & 144.6 & 0 & 144.6 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 144.6 & 144.6 & 144.6 & 144.6 & 0 & 144.6 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 36.1 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 36.1 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 36.1 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 36.1 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0094 & 0.0094 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix} \cdot 10^{-3} \text{ MBPTS}$$

$$\boldsymbol{i}^{\mathrm{UMTS}} = \begin{pmatrix}
25 & 1 & 2 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & \cdots \\
25 & 25 & 25 & 3 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & \cdots \\
25 & 25 & 25 & 25 & 4 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & \cdots \\
25 & 25 & 25 & 25 & 25 & 5 & 5 & 5 & 5 & 25 & 5 & 25 & \cdots \\
25 & 25 & 25 & 25 & 25 & 6 & 6 & 6 & 6 & 25 & 6 & 25 & \cdots \\
25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 9 & 25 & 25 & \cdots \\
25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 10 & 25 & 25 & \cdots \\
25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 11 & 25 & 25 & \cdots \\
25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 12 & 25 & 25 & \cdots \\
25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 22 & 22 & \cdots \\
25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & \cdots \\
25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & 25 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

$\boldsymbol{b}^{\mathrm{UMTS}} = (612, 612, 578, 578, 144.6, 144.6, 72.3, 72.3, 36.1, 36.1, 36.1, 36.1, 4.52, 4.52,$
$4.52, 4.52, 0.677, 0.677, 0.677, 0.226, 0.226, 0.0094, 0.0094, 0.0094, 0, 0, \ldots, 0) \cdot 10^{-3}$ MBPTS

Stage model

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

(b)

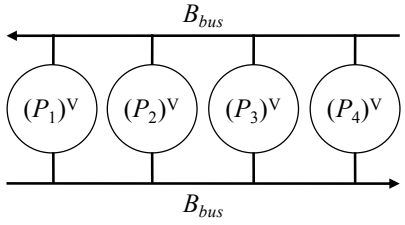
Fig. 3.7. UMTS task graph: graphical (a) and mathematical (b) modeling.

Table 3.12 Modeling parameters of the UMTS task graph of Fig. 3.7.

Computation	Communication	Communication-to-computation
$M(\text{UMTS}) = 24$	$con^{\text{UMTS}} = 0.123$	$ccr^{\text{UMTS}} = 0.302$ MBPTS / MOPTS
$(c_{\text{T}})^{\text{UMTS}} = 13.675$ MOPTS	$(b_{\text{T}})^{\text{UMTS}} = 4.135$ MBPTS	$\boldsymbol{x}^{\text{UMTS}}$ (see Fig. 3.8)
$(c_{\text{M}})^{\text{UMTS}} = 0.570$ MOPTS	$(b_{\text{M}})^{\text{UMTS}} = 0.122$ MBPTS	$(x_{\text{M}})^{\text{UMTS}} = 0.0729$
$(h_{\text{C}})^{\text{UMTS}} = 0.129$	$(h_{\text{B}})^{\text{UMTS}} = 0.0885$	$(x_{\text{STD}})^{\text{UMTS}} = 0.507$
	$(n_{\text{TS}})^{\text{UMTS}} = 17$	
	$g^{\text{UMTS}} = 17 / 24 = 0.708$	

$$\boldsymbol{x}^{\text{UMTS}} = \begin{pmatrix}
0 & 5.529 & 5.529 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
5.529 & 0 & 0 & 1.105 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
5.529 & 0 & 0 & 0 & 1.105 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\
0 & 1.105 & 0 & 0 & 0 & 0.126 & 0.126 & 0.126 & 0.126 & 0 & 0.326 & 0 & \cdots \\
0 & 0 & 1.105 & 0 & 0 & 0.126 & 0.126 & 0.126 & 0.126 & 0 & 0.326 & 0 & \cdots \\
0 & 0 & 0 & 0.126 & 0.126 & 0 & 0 & 0 & 0 & 0.051 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0.126 & 0.126 & 0 & 0 & 0 & 0 & 0.051 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0.126 & 0.126 & 0 & 0 & 0 & 0 & 0.051 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0.126 & 0.126 & 0 & 0 & 0 & 0 & 0.051 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0.051 & 0.051 & 0.051 & 0.051 & 0 & 0.001 & 0.003 & \cdots \\
0 & 0 & 0 & 0.326 & 0.326 & 0 & 0 & 0 & 0 & 0.001 & 0 & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.003 & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

Fig. 3.8. $\boldsymbol{x}^{\text{UMTS}}$.



(a)

$$\boldsymbol{I}^{\text{V}} = \begin{pmatrix} 1 & 5 & 5 & 5 \\ 6 & 2 & 5 & 5 \\ 6 & 6 & 3 & 5 \\ 6 & 6 & 6 & 4 \end{pmatrix}$$

$\boldsymbol{B}^{\text{V}} = (\infty, \infty, \infty, \infty, B_{bus}, B_{bus}, 0, \ldots, 0)$ MBPTS

(b)

$CON^{\text{V}} = 1 \qquad (B_{\text{T}})^{\text{V}} = 2 \cdot B_{bus}$

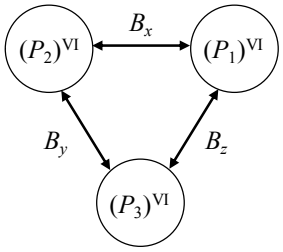
$(H_{\text{B}})^{\text{V}} = 0 \qquad (B_{\text{M}})^{\text{V}} = B_{bus}$

$$\boldsymbol{LF}^{\text{V}} = \begin{pmatrix} \infty & 0.5 & 0.5 & 0.5 \\ 0.5 & \infty & 0.5 & 0.5 \\ 0.5 & 0.5 & \infty & 0.5 \\ 0.5 & 0.5 & 0.5 & \infty \end{pmatrix}$$

$(LF_{\text{M}})^{\text{V}} = 0.5$

(c)

Fig. 3.9. SDR platforms V (a), its communications models (b) and parameters (c).



(a)

$$\boldsymbol{I}^{\text{VI}} = \begin{pmatrix} 1 & 4 & 5 \\ 4 & 2 & 6 \\ 5 & 6 & 3 \end{pmatrix}$$

$\boldsymbol{B}^{\text{VI}} = (\infty, \infty, \infty, B_x, B_y, B_z, 0, 0, 0)$ MBPTS

(b)

$CON^{\text{VI}} = 1 \qquad (B_{\text{T}})^{\text{VI}} = B_x + B_y + B_z$

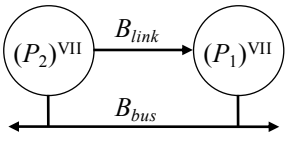
$(H_{\text{B}})^{\text{VI}} = 0 \qquad (B_{\text{M}})^{\text{VI}} = (B_x + B_y + B_z) / 3$

$$\boldsymbol{LF}^{\text{VI}} = \begin{pmatrix} \infty & B_x / B_{\text{T}} & B_z / B_{\text{T}} \\ B_x / B_{\text{T}} & \infty & B_y / B_{\text{T}} \\ B_z / B_{\text{T}} & B_y / B_{\text{T}} & \infty \end{pmatrix}$$

$(LF_{\text{M}})^{\text{VI}} = 1/3$

(c)

Fig. 3.10. SDR platforms VI (a), its communications models (b) and parameters (c).



(a)

$$\boldsymbol{I}^{\text{VII}} = \begin{pmatrix} 1 & ? \\ ? & 2 \end{pmatrix}$$

$\boldsymbol{B}^{\text{VII}} = (\infty, \infty, ?, ?)$ MBPTS

(b)

Fig. 3.11. A communication network (a) that cannot be captured by the proposed communication models (b).

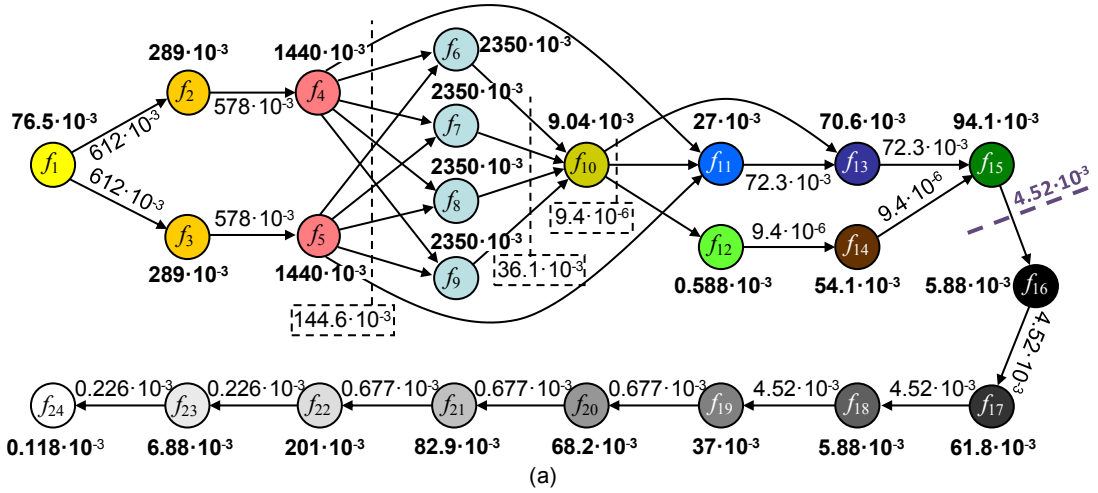
3.6 Additional Modeling Features and Extensions

3.6.1 Dividing and Merging of Platforms or Applications

The simplicity and modularity of our modeling approach facilitates an easy introduction of additional models and parameters. The mathematical and graphical models facilitate applying matrix processing and graph theoretic techniques, as will be shown in later chapters. A particular feature of the proposed models is that it eases the dividing and merging of platforms or applications.

The dividing of SDR applications is illustrated in Fig. 3.12 for the UMTS task graph of Fig. 3.7a and the data flow model $\boldsymbol{l}^{\mathrm{UMTS}}$. This approach generates two subgraphs, the first representing the chip-rate processing part of the UMTS receiver ($f_1, f_2, \ldots, f_{15}$) and the second containing the bit-rate processing modules ($f_{16}, f_{17}, \ldots, f_{24}$). The chip and bit-rate data flow requirements are then captured by a 15 x 15 and a 9 x 9 matrix. The two new matrices are joined through the data flow requirement between the *Rake Receiver* (f_{15}) and the *Physical Channel Demapping* (f_{16}). (The model $\boldsymbol{i}^{\mathrm{UMTS}}$ would be correspondingly split into two matrices of 15 x 15 and 9 x 9 elements, whereas $\boldsymbol{b}^{\mathrm{UMTS}}$ would be divided into two vectors of 225 and 81 elements.)

From Fig. 3.12 we infer that merging two SDR applications is the reverse process of the dividing procedure. Since computing resources and requirements are symmetrically modeled, SDR platforms can be equivalently merged or divided. When dividing an SDR platform or application, the connecting points, if any, need to be correctly managed. This means that the data flow between f_{15} and f_{16} in the example of Fig. 3.12 requires the allocation of $4.52 \cdot 10^{-3}$ MBPTS. Adding nodes with zero processing requirements (pseudonodes) could therefore be useful.



(a)

Chip-rate data flow demands

(15 x 15)

$\boldsymbol{l}^{\mathrm{UMTS}} =$

	1	2	⋯	13	14	15	16	17	⋯	23	24
1	0	0.612	0	0	0	0	0	0	0	0	0
2	0	0	0	0.578	0	0	0	0	0	0	0
⋮	0	0	⋱	⋮	⋮	⋮	⋮	⋮	⋰	0	0
13	0	0	⋯	0	0	$72 \cdot 10^{-3}$	0	0	⋯	0	0
14	0	0	⋯	0	0	$9.4 \cdot 10^{-6}$	0	0	⋯	0	0
15	0	0	⋯	0	0	0	$4.52 \cdot 10^{-3}$	0	⋯	0	0
16	0	0	⋯	0	0	0	0	$4.5 \cdot 10^{-3}$	⋯	0	0
17	0	0	⋯	0	0	0	0	0	⋯	0	0
⋮	0	0	⋰	⋮	⋮	⋮	⋮	⋮	⋱	0	0
23	0	0	0	0	0	0	0	0	0	0	$226 \cdot 10^{-6}$
24	0	0	0	0	0	0	0	0	0	0	0

(24) (24)

Bit-rate data flow demands

(9 x 9)

(b)

Fig. 3.12. Exemplifying the division of the UMTS task graph: graphically (a) and mathematically (b).

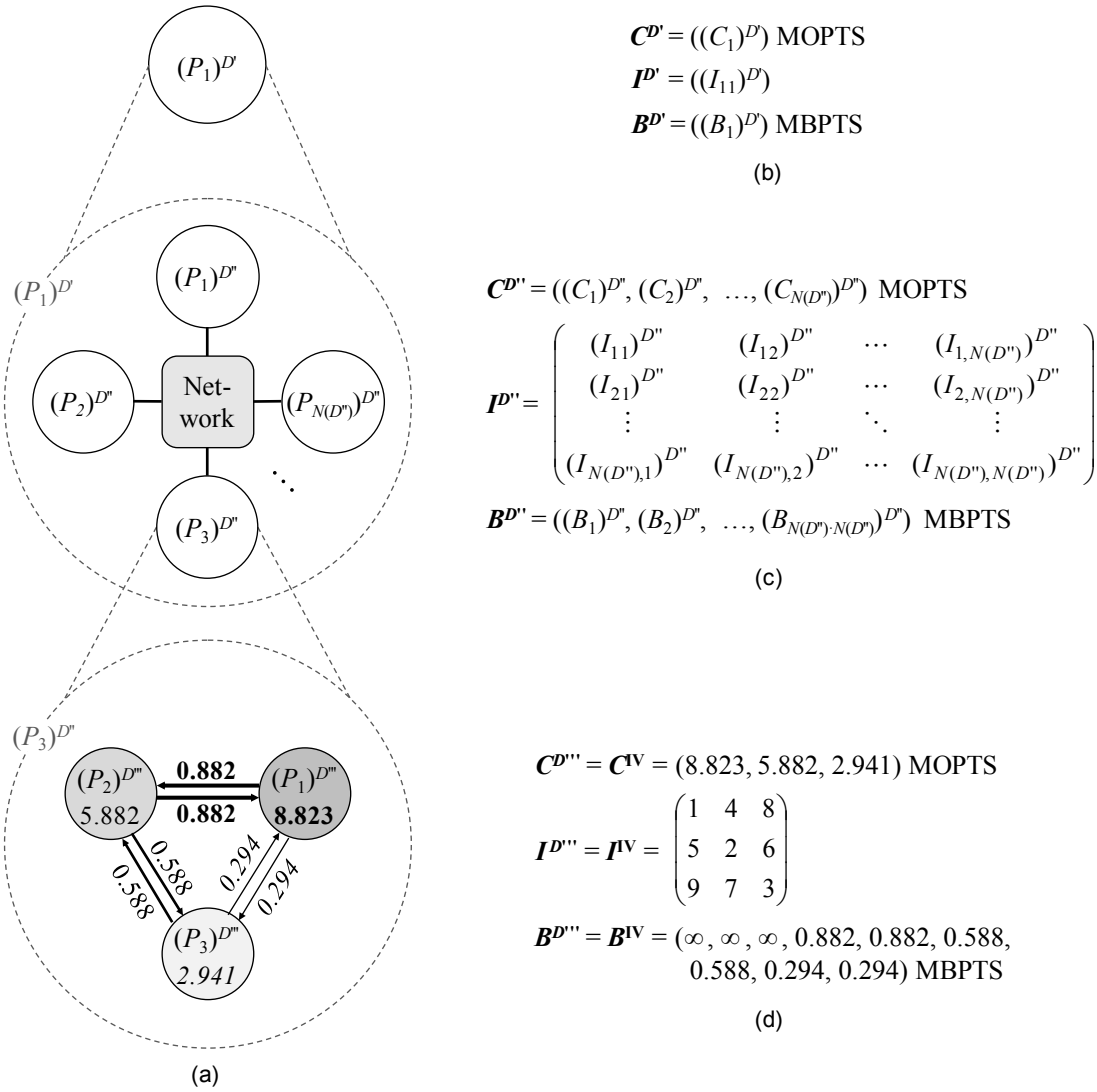


Fig. 3.13. Illustration of the hierarchical platform modeling: Three hierarchy levels (a) and the corresponding platform models (b-d).

These techniques may be practical for solving problems of different dimensions. It may, particularly, be inefficient or impossible to directly solve complex computing resource management problems. In case of multiuser base station, for instance, SDR applications could be individually mapped to the computing resources of distinct sub-cluster of the processing array. As sessions are dynamically initiated and terminated, this would ease the reuse of computing freed resources, being rather concentrated than widely distributed. From Fig. 3.12 finally follows that splitting an SDR application into two parts, approximately halves the memory requirement for storing the data flow demands.

3.6.2 Hierarchical Modeling

Clustering is a common technique for managing large arrays of processors, computational grids, or supercomputers. We consider it as another computing resource management facility. A hierarchical clustering is, particularly, useful in the context of this dissertation. We, therefore, discuss it briefly here while presenting the hierarchical modeling extension.

SDR platforms can be modeled at different hierarchy levels. The topmost level may be represented through a single processing unit (Fig. 3.13a), which abstracts an array of processors. The processing power at this level may then be the sum of processing powers of the underlying physical processors. The communication models would then be two 1 x 1 matrices $\boldsymbol{I}^{D'}$ and $\boldsymbol{B}^{D'}$ (Fig. 3.13b), indicating the total interprocessor communication capacity at another hierarchy level, for instance. At the next level we may have $N(D'')$ processors interconnected through some network (Fig. 3.13a and c). At the third level we

could find clusters of $N(D''') = 3$ processors (Fig. 3.13a and d). We consider clusters of three processors as the basic computing units in this dissertation.

We can correspondingly extend the application modeling to account for the three application levels of definitions 2, 3, and 4: SDR application level, SDR function level, and SDR process level. Another level on top of the SDR application level could join several SDR applications that need to be executed on a single SDR platform, such as an SDR-BS.

Hierarchical clustering may be useful for efficiently managing large arrays of processors. SDR-BSs, in particular, execute several SDR applications, typically one or more per user. Applying a hierarchical computing resource management based on the hierarchical modeling could then be as follows: The set of SDR applications assigned to $(P_1)^{D'}$ are distributed among the $N(D'')$ clusters, managing the processing resources $(C_1)^{D''}$ through $(C_{N(D'')})^{D''}$ and the available interprocessor bandwidths at hierarchy level 2. After finding a feasible allocation of computing resources to computing requirements, we can proceed with assigning computing resources to computing requirements at hierarchy level 3. At this level, SDR functions are distributed on the available processors.

The hierarchical modeling extension shows the flexibility of our modeling proposal. It, moreover, indicates how the particular study of this dissertation should be viewed within the overall SDR computing resource management context, consisting of SDR mobile terminals, base stations, and networks. This document does, though, not further discuss hierarchical modeling and its computing resource management implications.

3.7 Summary

We have presented a modular and flexible SDR computing system modeling, which consist of the platform and application modeling. Through instantiations of modeling templates, we can systematically capture all relevant platform and application characteristics, including the available computing resources and the corresponding requirements. Several parameters complement our modeling proposal. Each parameter points out some specific platform or application feature based on the introduced models. Chapters 5 and 6 will demonstrate the suitability of our modeling contribution.

This concludes the first part of our SDR computing resource management framework. The computing system modeling supports the computing resource management of Chapter 4. Nevertheless, the computing system modeling is independent of the computing resource management and has a specific function within the framework. The computing resource management algorithm does, for instance, not assume any particular interconnection network; it is the modeling that takes care of this. Hence, our framework can be applied to a wide variety of platforms and applications, a variety that characterizes SDR.

4

SDR Computing Resource Management

4.1 Introduction

Heterogeneous computing research has addressed many different mapping and scheduling problems and proposed a wide variety of specific solutions (Section 2.4). The SDR computing resource management problem, however, requires a more flexible approach (Section 2.5). We, therefore, suggest a more general computing resource management framework for the dynamic SDR computing and radio environments. Although our solution addresses the SDR computing resource management problem of Section 2.3 and assumes the SDR-specific system models of Chapter 3, it is also applicable to other real-time computing contexts [144].

This chapter presents a modular and open computing resource management approach. It distinguishes between the algorithm and the objective or cost function. This modularity (Fig. 4.1) increases the flexibility of applying different cost functions, facilitating a dynamic adjustment or exchange of the computing resource management policy as a function of the momentary environmental conditions. One policy would be meeting real-time computing constraints in hard to meet conditions and another, optimizing the energy consumption. Different environmental conditions represent different computing resource management scenarios or problems, which possibly require specific solutions. This is in line with Lee and Aggarwal [38], who concluded that "different objective functions and mapping algorithms are to be adopted for different applications", where *application* in the context of the citation refers to scenario or problem.

The computing resource management problem that this dissertation tackles can be formulated as a general mapping problem: A precedence-constrained task graph (with computing requirements) is to be mapped to a heterogeneous multiprocessor platform (with limited computing resources). With N processors and M tasks, there are N^M different mapping options. Not all solutions are necessarily feasible and only one or a few are optimal in some sense, whether minimizing the makespan or pursuing another objec-

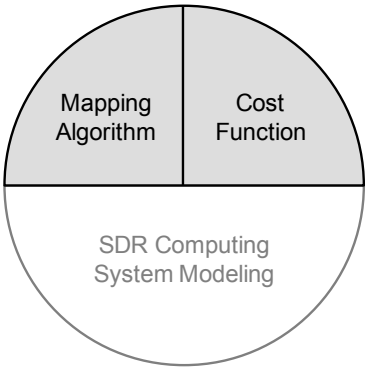


Fig. 4.1. Modular SDR computing resource management.

tive. An optimal and even a feasible solution to nontrivial multiprocessor mapping or scheduling problems is generally hard to find and may require an exhaustive search [58], [64]. It has been shown that many related (optimization) problems are NP-complete [37], [47], [82], [83], [84], [85]. This means that the execution time for finding an optimal solution can become extremely long and impractical for any realistic problem size.

Under the above assumptions, we suggest a heuristic approach that systematically maps computing requirements to computing resources. It should permit dynamic parameter or algorithm adjustments as well as facilitate the application of pre and postprocessing add-ons. The t_w- and g_w-mapping (Section 4.2) are two such approaches. They are dynamically adjustable through parameter w─the window size─which controls the performance versus complexity tradeoff. Both algorithms permit the application of different cost functions, which guide the mapping process. We introduce a cost function that manages the limited computing resources as a function of the hard real-time computing requirements (Section 4.3). We analytically analyze the algorithms' processing complexities (Section 4.4) and conclude this chapter with an SDR computing resource management example (Section 4.5).

The mapping algorithms can, in principle, handle any SDR platform or application. The cost function, which actually manages the computing resources and requirements, assumes the modeling of Chapter 3. Except for Section 4.5, this chapter uses the modeling expressions of Chapter 3 without SDR platform index D and SDR application index d for improving the readability.

4.2 Mapping Algorithms

It is not feasible to adapt previously introduced algorithms to the SDR computing resource management context and to evaluate their performance within our framework [49]. These algorithms principally lack the flexibility that the highly dynamic SDR environment requires. We therefore introduce the t_w-mapping (Section 4.2.1), a flexible mapping algorithm based on the dynamic programming principles [146]. We also implement an extended greedy approach, the g_w-mapping (Section 4.2.2), which constitutes the reference mapping algorithm for performance evaluations (Chapter 5).

The following concepts facilitate a formal algorithm description. The t_w-mapping diagram contains a matrix of N x M (row x column) t-nodes (Fig. 4.2). 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 connects to all t-nodes at *step* i+1. The sequence of processors, $[P_{k(0)}\ P_{k(1)}\ \cdots\ P_{k(w)}]_i$, identifies the w-path, a path of length w, that 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 $P_{k(w)}$ 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. 4.2. It is associated with t-node $\{P_1, f_2\}$. Table 4.1 summarizes the most important variables and expressions that facilitate formal algorithm presentations.

Since both algorithms are cost function independent, their formal descriptions (Sections 4.2.1 through 4.2.2) do not assume a particular cost function. Section 4.2.3, on the other hand, provides an introduction to the t_w- and g_w-mapping by means of a simple mapping problem and a specific cost function.

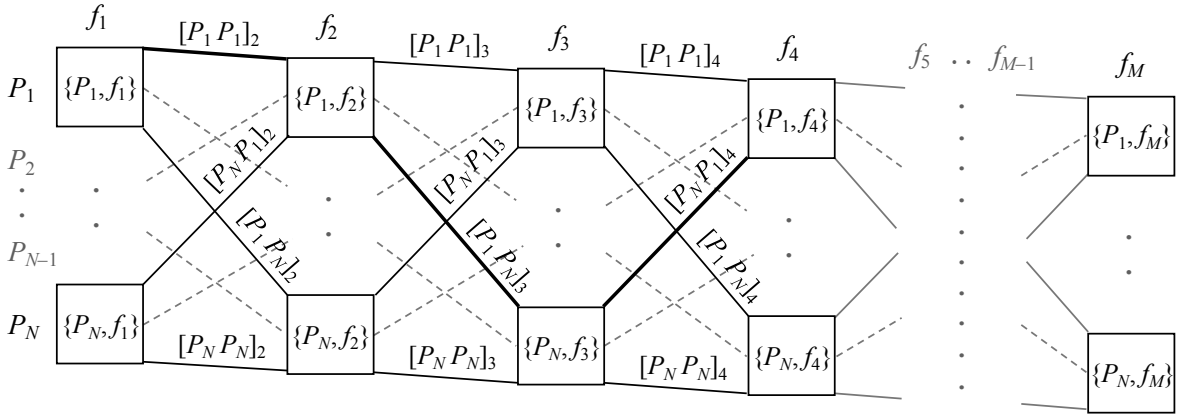


Fig. 4.2. The t_w-mapping diagram.

Table 4.1 Algorithm-specific parameters and expressions.

Parameter or expression	Range (argument range)	Description
N	$\mathbb{N}$	number of processors
M	$\mathbb{N}$	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 indicating 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 $\{P_{k(1)}, f_i\}$
$[P_{k(l-1)}\ P_{k(l)}]_h$	($l \neq 0$; $h \in 2, 3, \ldots, M$)	edge, or 1-path, between t-nodes $\{P_{k(l-1)}, f_{h-1}\}$ and $\{P_{k(l)}, f_h\}$
$\mathrm{WT}[P_{k(l-1)}\ P_{k(l)}]_h$	($l \neq 0$; $h \in 2, 3, \ldots, M$)	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 up to t-node $\{P_{k(w)}, f_{i+w-1}\}$ on 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 $\{P_{k(l)} f_i\}$
$P(f_j)$	$P_1, P_2, \ldots, P_N$; $(f_1, f_2, \ldots, f_M)$	processor allocated to SDR function f_j

4.2.1 The t_w-mapping

The t_w-mapping algorithm (Fig. 4.3) can be divided into three parts: *processing at step 1* (lines 1 and 2 in Fig. 4.3), *processing at steps i = 2, 3, ..., M–w+1* (lines 3-18), and *postprocessing* (lines 19 and 20).

A) 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 all M SDR functions have been processed, or *premapped*.

B) Part II: Processing at Steps i = 2, 3, ..., 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 $[P_1\ P_{k(1)}]_i$, $[P_2\ P_{k(1)}]_i$, …, 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 costs as

$$\mathrm{CT}[P_{k(0)}\ P_{k(1)}]_i = \mathrm{CT}\{P_{k(0)}, f_{i-1}\} + \mathrm{WT}[P_{k(0)}\ P_{k(1)}]_i \tag{4.1}$$

(line 12, bold). Edge $[P_{k(0)*}\ P_{k(1)}]_i$ is obtained from

$$k(0)* = \underset{k(0)=1,2,\ldots,N}{\arg\min} \left\{ \mathrm{CT}[P_{k(0)}\ P_{k(1)}]_i \right\} \tag{4.2}$$

and represents the premapping decision at t-node $\{P_{k(1)}, f_i\}$. (The function $\mathrm{argmin}\{x\}$ 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

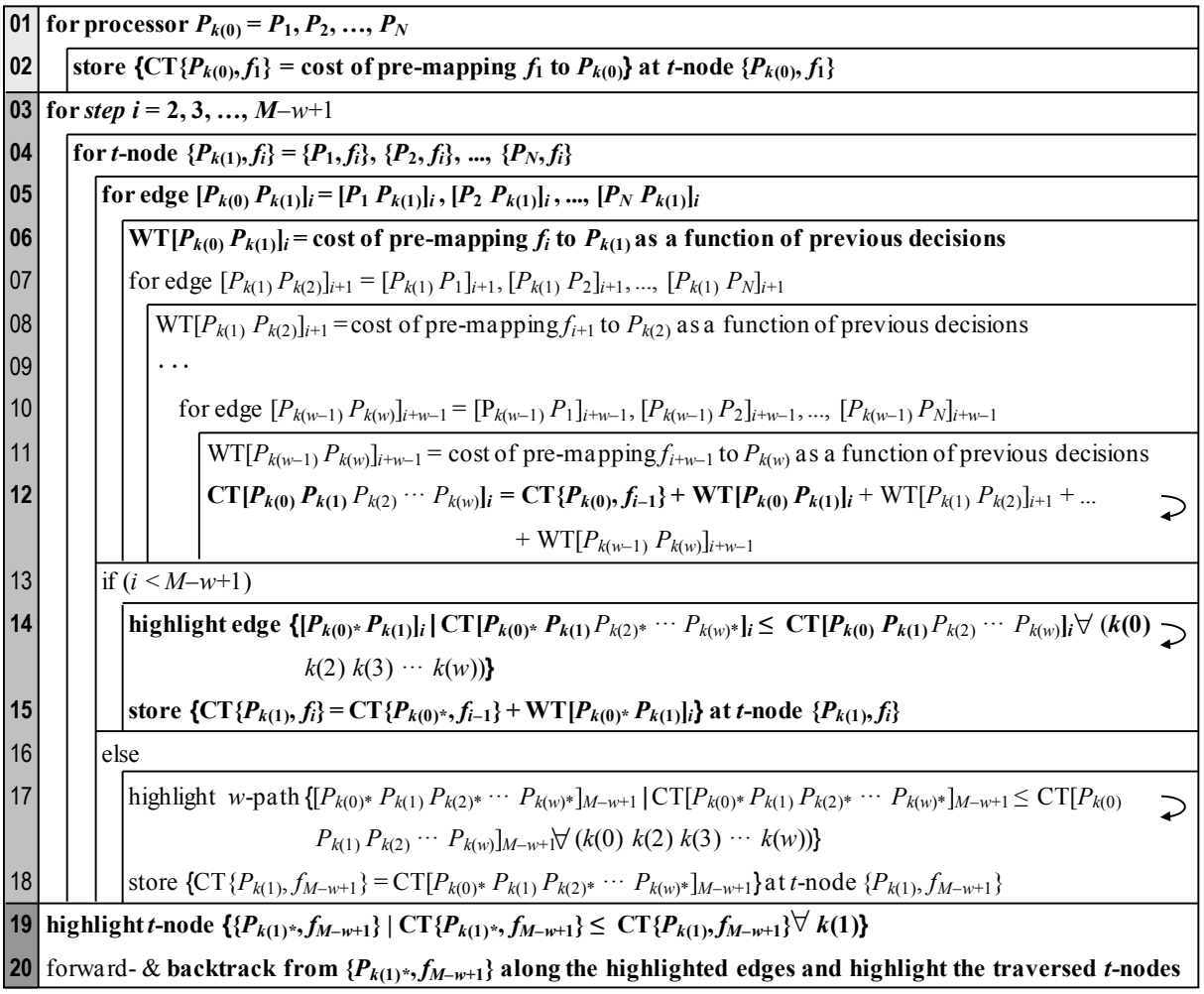


Fig. 4.3. t_w-mapping─pseudocode (the bold expressions describe the t_1-mapping).

$$\{k(0)^*, k(2)^*, k(3)^*, \ldots, k(w)^*\} = \underset{k(l)=1,2,\ldots,N;\forall l \neq 1}{\arg\min} \left\{ CT[P_{k(0)}\ P_{k(1)}\ P_{k(2)}\ \cdots\ P_{k(w)}]_i \right\}, \tag{4.3}$$

which returns the indices of the w-path that has the minimum accumulated cost. In 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 $[P_{k(0)^*}\ P_{k(1)}\ P_{k(2)^*} \cdots P_{k(w)^*}]_{M-w+1}$ 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 ($w \geq 1$) proceeds with *step* i+1 in the case where $i < M-w+1$ and with part III otherwise (line 3).

C) Part III: Postprocessing

Part III of the algorithm postprocesses the premapping decisions of parts I and II. The t_w-mapping ($w \geq 1$) 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 of $w > 1$) 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.

D) Parameter w

The window size w can take any integer value between 1 and M–1 (Table 4.1). It 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 results in N (partially) different mapping options (bold expressions in Fig. 4.3).

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) before selecting the path of minimum cost (lines 17 and 18). After processing all N t-nodes at *step* 2 (line 4), the algorithm highlights t-node $\{P_{k(1)^*}, f_2\}$ (line 19) and forwards and backtracks the t_w-mapping diagram, obtaining the final mapping (line 20). It finds an optimal solution for the given problem and cost function.

The window size also controls the tradeoff between computing efficiency and mapping performance: The higher the window size, the higher the algorithm's complexity (Section 4.4), but the better the mapping results (Section 4.5 and Chapters 5 and 6).

4.2.2 The g_w-mapping

The g_w-mapping algorithm (Fig. 4.4) consists of two parts, the *processing at step 1* (lines 1-3 in Fig. 4.4) and the *processing at steps i = 2, 3, …, M–w+1* (lines 4-18). It is a forward processing algorithm. That is, $P(f_i) = P_1, P_2, \ldots$, or P_N is firmly specified as the processor to be allocated to SDR function f_i at processing *step i*. The algorithm therefore proceeds as follows.

A) Part I: Processing at Step 1

The g_w-mapping's part I computes the costs of mapping f_1 to each one of the N processor (lines 1 and 2). It then selects the processor to be allocated to f_1 as

$$P(f_1) = \{P_{k(0)^*} \,|\, \mathrm{CT}\{P_{k(0)^*}, f_1\} \le \mathrm{CT}\{P_{k(0)}, f_1\} \;\forall\; k(0)\}, \tag{4.4}$$

while discarding all but the active t-node at *step* 1, t-node $\{P(f_1), f_1\}$ (line 3).

B) Part II: Processing at Steps i = 2, 3, …, M–w+1

In part II, the g_1-mapping analyzes the N outgoing edges from the active t-node at *step i*–1 (lines 5, 6, and 12 bold). It solves

$$k(1)^* = \underset{k(1)=1,2,\ldots,N}{\arg\min} \left\{ \mathrm{CT}[P(f_1) \;\; P_{k(1)}]_i \right\} \tag{4.5}$$

for finding the minimum-cost edge $[P(f_{i-1}) \;\; P_{k(1)^*}]_i$ and, thus, the processor to be allocated to f_i , $P(f_i) = P_{k(1)^*}$. The active t-node at step i becomes $\{P(f_i), f_i\}$ (line 14). It retains the accumulated mapping cost (line 15). If $i < M$, the g_1-mapping proceeds with *step i*+1 (line 4).

The g_w-mapping ($w > 1$) examines N^w w-paths (lines 5-12). Each of these w-paths originates at t-node $\{P(f_{i-1}), f_{i-1}\}$, runs through a t-node at *step i*, through a t-node at *step i*+1, …, and terminates at a t-node at *step i*+w–1. The algorithm then solves

$$\{k(1)^*, k(2)^*, k(3)^*, \ldots, k(w)^*\} = \underset{k(l)=1,2,\ldots,N;\forall l\neq 0}{\arg\min} \left\{ \mathrm{CT}[P(f_{i-1}) \;\; P_{k(1)} \;\; P_{k(2)} \;\cdots\; P_{k(w)}]_i \right\}, \tag{4.6}$$

obtaining the indices of the minimum-cost w-path $[P(f_{i-1}) \; P_{k(1)^*} \; P_{k(2)^*} \cdots P_{k(w)^*}]_i$.

If $i < M$–w+1, the g_w-mapping identifies $P(f_i) = P_{k(1)^*}$ as the processor to be allocated to SDR function f_i, highlighting the corresponding t-node and storing the accumulated mapping cost up to this node (lines 13-15). *Step i*+1 then follows (line 4). If $i = M$–w+1, w-path $[P(f_{i-1}) \; P_{k(1)^*} \; P_{k(2)^*} \cdots P_{k(w)^*}]_i$ specifies the remaining processor allocations: $P(f_{M-w+1}) = P_{k(1)^*}$, $P(f_{M-w+2}) = P_{k(2)^*}$, …, $P(f_M) = P_{k(w)^*}$ (lines 16 and 17). The total mapping cost,

$$\mathrm{CT}\{P(f_{M-w+1}), f_{M-w+1}\} = \mathrm{CT}[P(f_{M-w}) \; P(f_{M-w+1}) \; P(f_{M-w+2}) \cdots P(f_M)]_{M-w+1}, \tag{4.7}$$

is finally stored at t-node $\{P(f_{M-w+1}), f_{M-w+1}\}$ (line 18).

C) Parameter w

Parameter w here establishes the decision window for part II of the g_w-mapping: To allocate a processor to SDR function f_i, the g_w-mapping computes all N^w different mapping combinations of the set of SDR functions between f_i and f_{i+w-1} to the N processors P_1 to P_N. The g_1-mapping thus chooses the best immediate allocation for each SDR function (as a function of the previous decisions); it is a greedy algorithm [145].

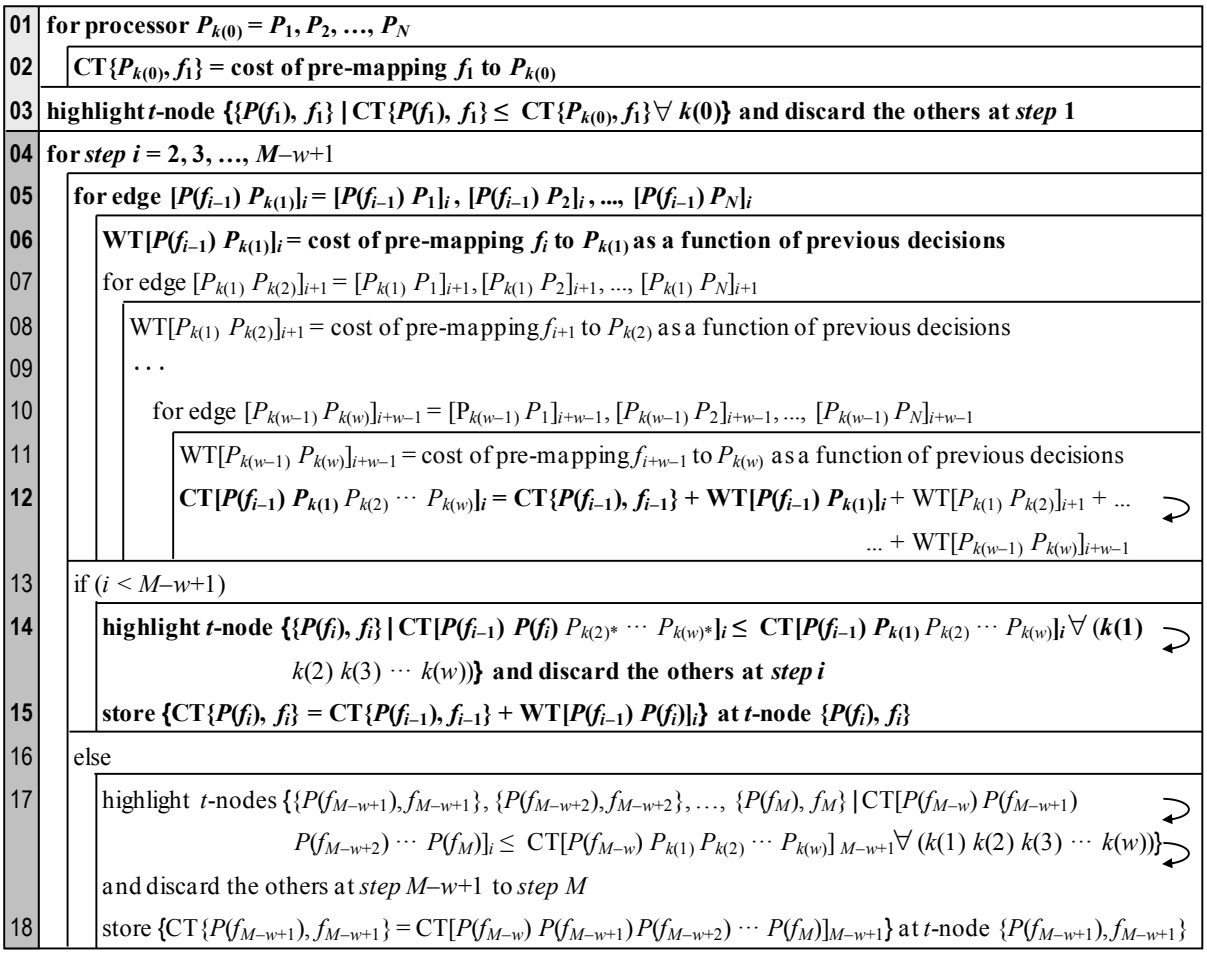


Fig. 4.4. g_w-mapping─pseudocode (the bold expressions describe the g_1-mapping).

The g_{M-1}-mapping, on the other hand, optimally solves the mapping of SDR functions f_2 to f_M with respect to the mapping of f_1. More precisely, after finding the local best match for SDR function f_1, the g_{M-1}-mapping performs an extensive search through the complete solution space for mapping the remaining $M-1$ SDR function to the N processors.

4.2.3 Exemplifying the t_w- and g_w-mapping

We consider a simple mapping problem for exemplifying the two mapping algorithms: The task graph of Fig. 4.5a is to be mapped to the platform model of Fig. 4.5b. The following figures illustrate how the t_w- and g_w-mapping solve this mapping problem for different window sizes. Part I is independent of w and is illustrated once for the t_w-mapping (Section 4.2.3A)) and once for the g_w-mapping (Section 4.2.3E)).

The tables, which are embedded within the figures, contain the premapping costs. These costs are due to cost function (4.9), which will be presented in Section 4.3.2. It is irrelevant here how the cost figures are actually obtained. It is, though, important to understand which paths need to be evaluated and how the decisions are made given the costs of these paths. Section 4.5 will discuss another mapping example, focusing on the cost calculations and resource updates.

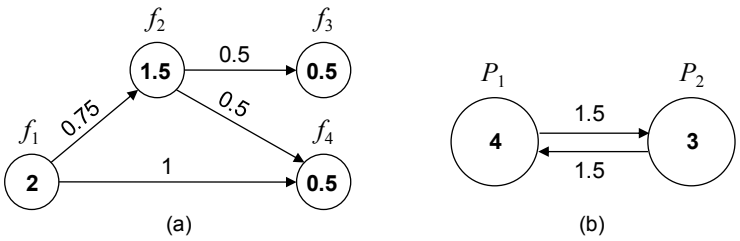


Fig. 4.5. A simple mapping problem: task graph (a) and platform model (b).

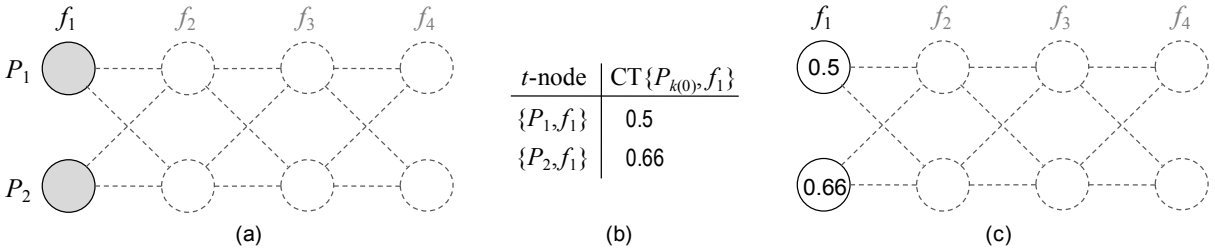
A) t_w-mapping*, Part I*

The t_w-mapping, part I premaps SDR function f_1 to each one of the N processors. It examines the $N = 2$ t-nodes $\{P_1, f_1\}$ and $\{P_2, f_1\}$ (Fig. 4.6a) and computes the two premapping costs (Fig. 4.6b), which update the t_w-mapping diagram (Fig. 4.6c). Fig. 4.6c is the basis for part II of the t_1-, t_2-, …, and t_{M-1}-mapping. Since $M = 4$, the maximum window size is $w = M–1 = 3$ (Section 4.2).

B) t_1-mapping*, Parts II and III*

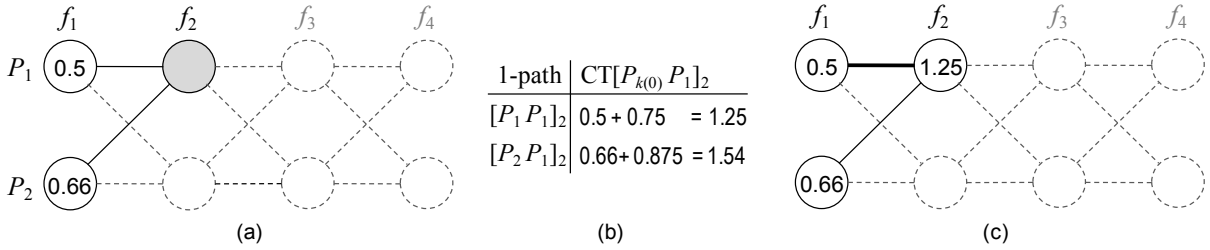
Part II of the t_1-mapping sequentially processes the t-nodes between *step* 2 and *step M*. At *step* 2, the ingoing edges associated with t-nodes $\{P_1, f_2\}$ and $\{P_2, f_2\}$ are examined, as Fig. 4.7a and Fig. 4.8a illustrate. The costs of the $N = 2$ edges per t-node are computed (Fig. 4.7b and Fig. 4.8b) and compared for deciding on the surviving edge (Fig. 4.7c and Fig. 4.8c). These decisions, $[P_1\ P_1]_2$ and $[P_1\ P_2]_2$, are the basis for the corresponding processing at *step* 3 (Fig. 4.9 and Fig. 4.10) and *step* 4 (Fig. 4.11 and Fig. 4.12).

Once finished with the forward processing, or part II of the algorithm, the t_1-mapping selects the minimum-cost t-node at step $M = 4$ (Fig. 4.13a) and backtracks the t_w-mapping diagram along the highlighted edges (Fig. 4.13b). The traversed t-nodes specify the complete mapping solution (Fig. 4.13c). The cost of this mapping is $\mathrm{CT}\{P_1, f_4\} = 2.75$.



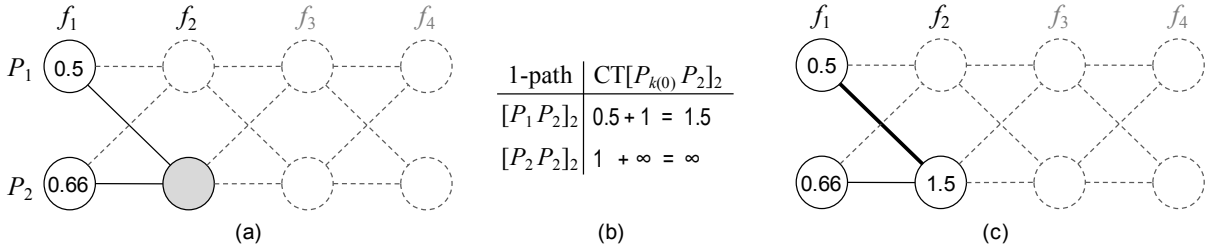
t-node	$\mathrm{CT}\{P_{k(0)}, f_1\}$
$\{P_1, f_1\}$	0.5
$\{P_2, f_1\}$	0.66

Fig. 4.6. t_w-mapping, part I: examined t-nodes (a), their costs (b), and the t_w-mapping diagram update (c).



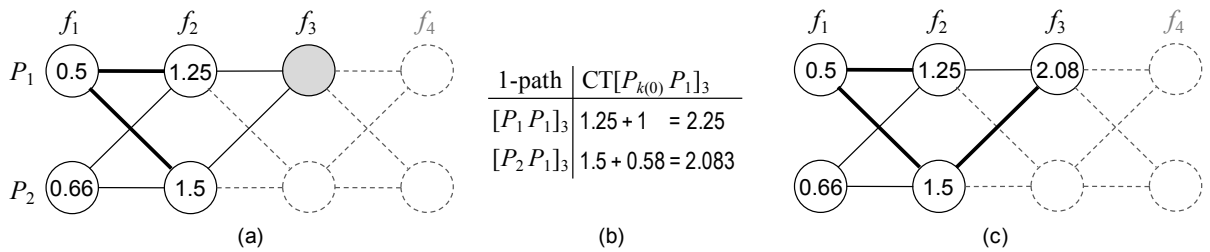
1-path	$\mathrm{CT}[P_{k(0)}\ P_1]_2$
$[P_1\ P_1]_2$	0.5 + 0.75 = 1.25
$[P_2\ P_1]_2$	0.66 + 0.875 = 1.54

Fig. 4.7. t_1-mapping, part II: examined edges at t-node $\{P_1, f_2\}$ (a), their costs (b), and the decision (c).



1-path	$\mathrm{CT}[P_{k(0)}\ P_2]_2$
$[P_1\ P_2]_2$	0.5 + 1 = 1.5
$[P_2\ P_2]_2$	1 + ∞ = ∞

Fig. 4.8. t_1-mapping, part II: examined edges at t-node $\{P_2, f_2\}$ (a), their costs (b), and the decision (c).



1-path	$\mathrm{CT}[P_{k(0)}\ P_1]_3$
$[P_1\ P_1]_3$	1.25 + 1 = 2.25
$[P_2\ P_1]_3$	1.5 + 0.58 = 2.083

Fig. 4.9. t_1-mapping, part II: examined edges at t-node $\{P_1, f_3\}$ (a), their costs (b), and the decision (c).

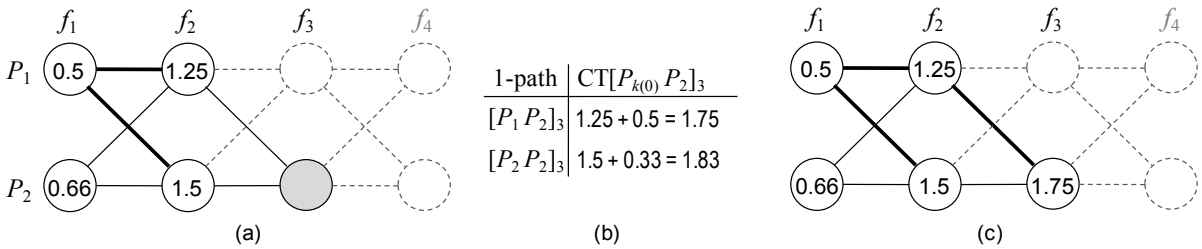


Fig. 4.10. t_1-mapping, part II: examined edges at t-node $\{P_2, f_3\}$ (a), their costs (b), and the decision (c).

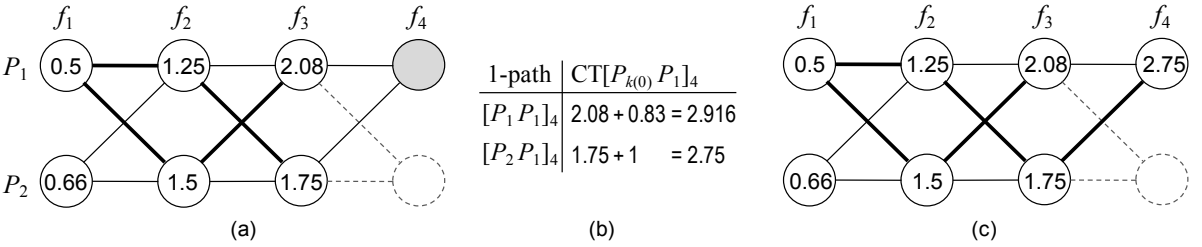


Fig. 4.11. t_1-mapping, part II: examined edges at t-node $\{P_1, f_4\}$ (a), their costs (b), and the decision (c).

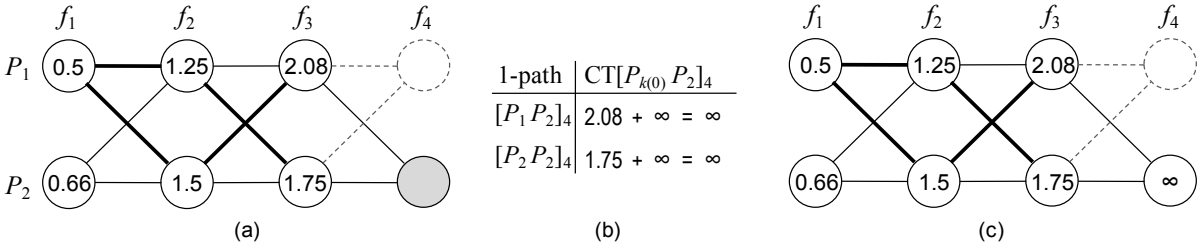


Fig. 4.12. t_1-mapping, part II: examined edges at t-node $\{P_2, f_4\}$ (a), their costs (b), and the decision (c).

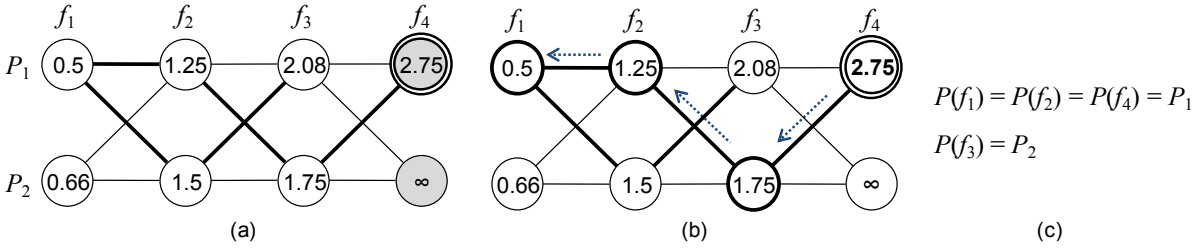


Fig. 4.13. t_1-mapping, part III: minimum-cost t-node selection (a), backtracking (b), and the final mapping solution (c).

C) t_2-mapping, Parts II and III

The t_2-mapping, part II examines $N^w = 4$ w-paths at each t-node of *step* 2 (Fig. 4.14a and Fig. 4.15a). Individually for each t-node, the accumulated costs of the corresponding 2-paths are computed and compared for t-node $\{P_1, f_2\}$ (Fig. 4.14b) and t-node $\{P_2, f_2\}$ (Fig. 4.15b). The minimum-cost 2-path specifies the single edge to be highlightes. Edges $[P_2\ P_1]_2$ and $[P_1\ P_2]_2$ are highlighted here (Fig. 4.14c and Fig. 4.15c) since pertaining to the minimum-cost 2-paths $[P_2\ P_1\ P_1]_2$ and $[P_1\ P_2\ P_1]_2$ (Fig. 4.14b and Fig. 4.15b), respectively.

The t-nodes at *step* 3 are correspondingly processed except for highlighting the minimum-cost w-paths entirely. These are the w-paths $[P_1\ P_1\ P_2]_3$ (Fig. 4.16b) and $[P_2\ P_2\ P_1]_3$ (Fig. 4.17b). The associated accumulated costs are finally stored at t-nodes $\{P_1, f_3\}$ (Fig. 4.16c) and $\{P_2, f_3\}$ (Fig. 4.17c).

Part III of the t_2-mapping then chooses the t-node of minimum cost at *step* 3, t-node $\{P_2, f_3\}$ in this case (Fig. 4.18a). The forward and backtracking process thus starts at t-node $\{P_2, f_3\}$ (Fig. 4.18b) and finds the final mapping (Fig. 4.18c). The cost of this mapping is CT$\{P_2, f_3\}$ = 2.416.

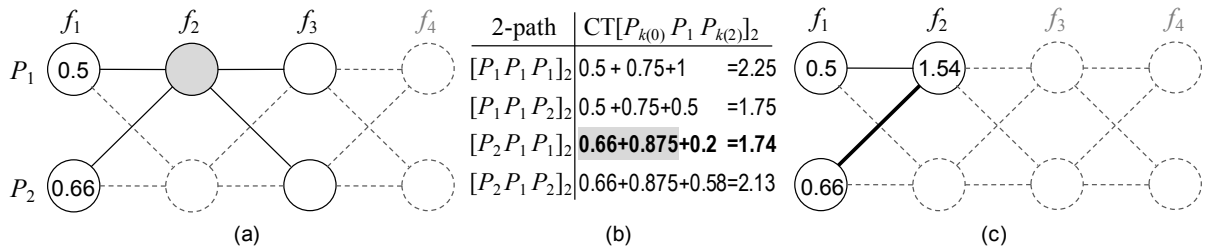


Fig. 4.14. t_2-mapping, part II: examined 2-paths at t-node $\{P_1, f_2\}$ (a), their costs (b), and the decision (c).

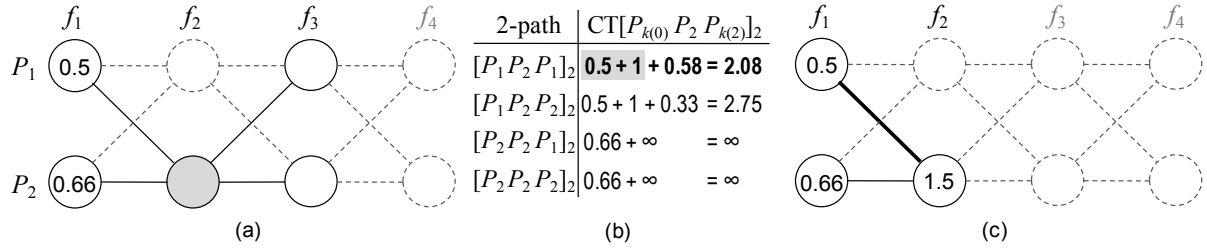


Fig. 4.15. t_2-mapping, part II: examined 2-paths at t-node $\{P_2, f_2\}$ (a), their costs (b), and the decision (c).

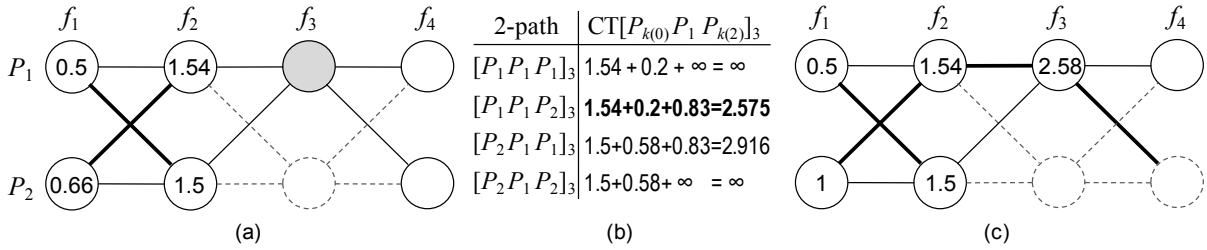


Fig. 4.16. t_2-mapping, part II: examined 2-paths at t-node $\{P_1, f_3\}$ (a), their costs (b), and the decision (c).

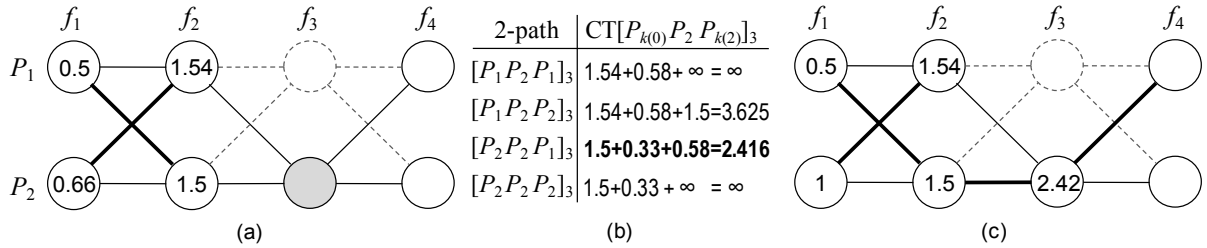


Fig. 4.17. t_2-mapping, part II: examined 2-paths at t-node $\{P_2, f_3\}$ (a), their costs (b), and the decision (c).

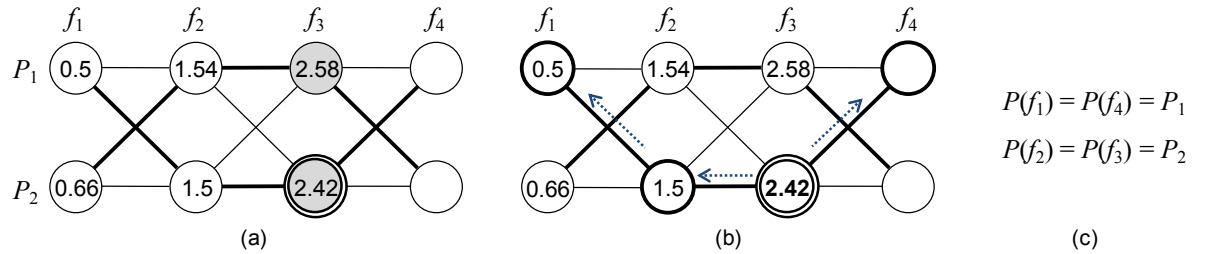
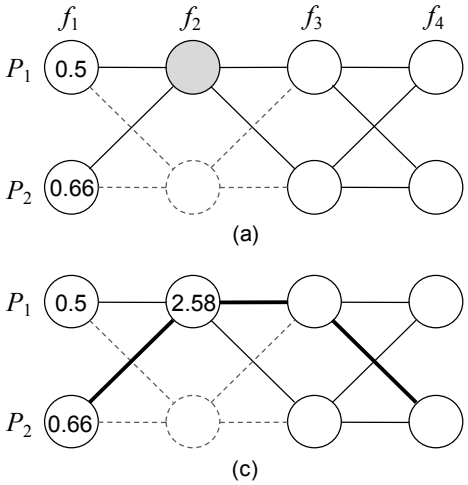


Fig. 4.18. t_2-mapping, part III: minimum-cost t-node selection (a), forward and backtracking (b), and the final mapping solution (c).

D) t_3-mapping, Parts II and III

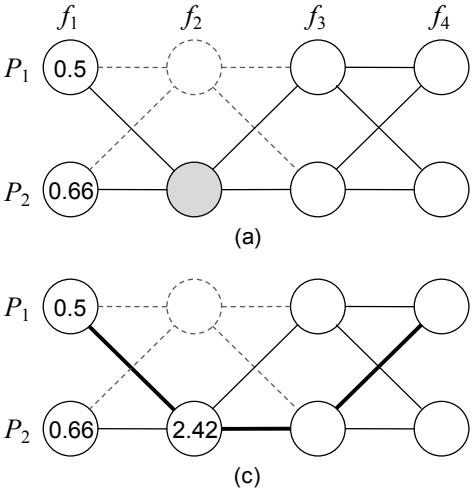
The t_3-mapping, part II examines $N^w = 8$ w-paths at each t-node of *step* 2 (Fig. 4.19a and Fig. 4.20a). For each of these t-nodes, the costs of the eight corresponding 3-paths are computed and compared (Fig. 4.19b and Fig. 4.20b). The 3-paths of minimum costs─$[P_2\ P_1\ P_1\ P_2]_2$ (Fig. 4.19b) and $[P_1\ P_2\ P_2\ P_1]_2$ (Fig. 4.20b)─are then highlighted (Fig. 4.19c and Fig. 4.20c).



3-path	$CT[P_{k(0)} P_1 P_{k(2)} P_{k(3)}]_2$	
$[P_1 P_1 P_1 P_1]_2$	0.5 + 0.75 + 1 + ∞	= ∞
$[P_1 P_1 P_1 P_2]_2$	0.5 + 0.75 + 1 + 1.833	= 4.083
$[P_1 P_1 P_2 P_1]_2$	0.5 + 0.75 + 0.5 + 1	= 2.75
$[P_1 P_1 P_2 P_2]_2$	0.5 + 0.75 + 0.5 + ∞	= ∞
$[P_2 P_1 P_1 P_1]_2$	0.66 + 0.875 + 0.2 + ∞	= ∞
$[P_2 P_1 P_1 P_2]_2$	**0.66 + 0.875 + 0.2 + 0.833**	**= 2.575**
$[P_2 P_1 P_2 P_1]_2$	0.66 + 0.875 + 0.833 + ∞	= ∞
$[P_2 P_1 P_2 P_2]_2$	0.66 + 0.875 + 0.833 + 1.5	= 3.875

(b)

Fig. 4.19. t_3-mapping, part II: examined 3-paths at t-node $\{P_1, f_2\}$ (a), their costs (b), and the decision (c).



3-path	$CT[P_{k(0)} P_2 P_{k(2)} P_{k(3)}]_2$	
$[P_1 P_2 P_1 P_1]_2$	0.5 + 1 + 0.583 + 0.833	= 2.916
$[P_1 P_2 P_1 P_2]_2$	0.5 + 1 + 0.583 + ∞	= ∞
$[P_1 P_2 P_2 P_1]_2$	**0.5 + 1 + 0.33 + 0.583**	**= 2.416**
$[P_1 P_2 P_2 P_2]_2$	0.5 + 1 + 0.33 + ∞	= ∞
$[P_2 P_2 P_1 P_1]_2$	0.66 + ∞	= ∞
$[P_2 P_2 P_1 P_2]_2$	0.66 + ∞	= ∞
$[P_2 P_2 P_2 P_1]_2$	0.66 + ∞	= ∞
$[P_2 P_2 P_2 P_2]_2$	0.66 + ∞	= ∞

(b)

Fig. 4.20. t_3-mapping, part II: examined 3-paths at t-node $\{P_2, f_2\}$ (a), their costs (b), and the decision (c).

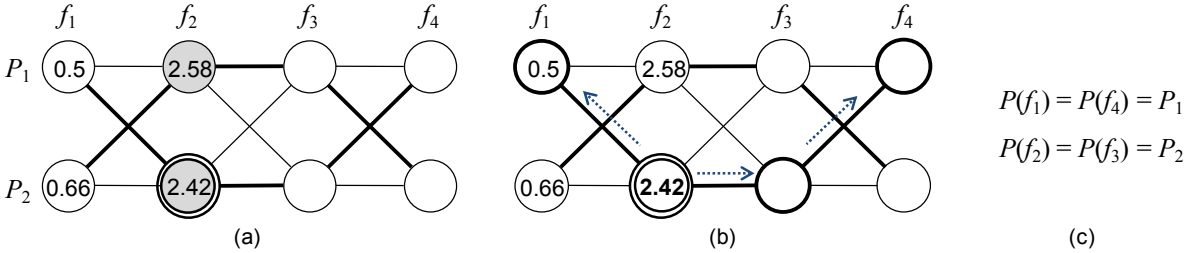


Fig. 4.21. t_3-mapping, part III: minimum-cost t-node selection (a), forward and backtracking (b), and the final mapping solution (c).

Part III finally finds the minimum-cost t-node at *step* 2, which is $\{P_2, f_2\}$ due to Fig. 4.21a, and traverses the trellis forward and backward (Fig. 4.21b) for obtaining the final mapping (Fig. 4.21c). The mapping cost is $CT\{P_2, f_2\} = 2.416$ (Fig. 4.21b), which is optimal for the given problem and cost function since testing all $N^M = 16$ mapping combinations of $M = 4$ SDR functions to $N = 2$ processors.

E) g_w-mapping, *Part I*

The g_w-mapping, part I examines the $N = 2$ t-nodes $\{P_1, f_1\}$ and $\{P_2, f_1\}$ (Fig. 4.22a). It computes the cost of mapping f_1 to P_1 and that of mapping f_1 to P_2 (Fig. 4.22b) before deciding which t-node to maintain and which to discard (Fig. 4.22c). The processor allocation $P(f_1) = P_1$ is the basis for part II of the g_w-mapping process.

F) g_1-mapping, *Part II*

At step 2, the g_1-mapping examines the $N^w = 2$ edges $[P_1\ P_1]_2$ and $[P_1\ P_2]_2$, which originate at the active *t*-node at *step* 1 (Fig. 4.23a). The corresponding accumulated costs are computed and compared (Fig. 4.23b) before the decision is made (Fig. 4.23c). The decision, *t*-node $\{P_1, f_2\}$, is the basis for the processing that follows, which is illustrated in Fig. 4.24 and Fig. 4.25. The decisions are *t*-node $\{P_2, f_3\}$ at *step* 3 (Fig. 4.24c) and *t*-node $\{P_1, f_4\}$ at *step* 4 (Fig. 4.25c). The complete mapping solution is then $P(f_1) = P(f_2) = P(f_4) = P_1$ and $P(f_3) = P_2$. The cost of this mapping is $\mathrm{CT}\{P_1, f_4\} = 2.75$ (Fig. 4.25c).

G) g_2-mapping, *Part II*

Part II of the g_2-mapping proceeds as follows: $N^w = 4$ *w*-paths are examined at *step* 2 (Fig. 4.26a). Any of these *w*-paths originate at *t*-node $\{P_1, f_1\}$, the active *t*-node at *step* 1 due to Fig. 4.22c. The accumulated costs associated with these *w*-paths are computed and compared (Fig. 4.26b). Based on these costs, *t*-node $\{P_1, f_2\}$ becomes the active *t*-node at step 2, whereas *t*-node $\{P_2, f_2\}$ is discarded (Fig. 4.26c). $\mathrm{CT}\{P_1, f_2\} = 1.25$ is stored at *t*-node $\{P_1, f_2\}$ (Fig. 4.26c). It corresponds to the mapping cost of allocating f_1 and f_2 to P_1. The processing at *step* 3 then follows.

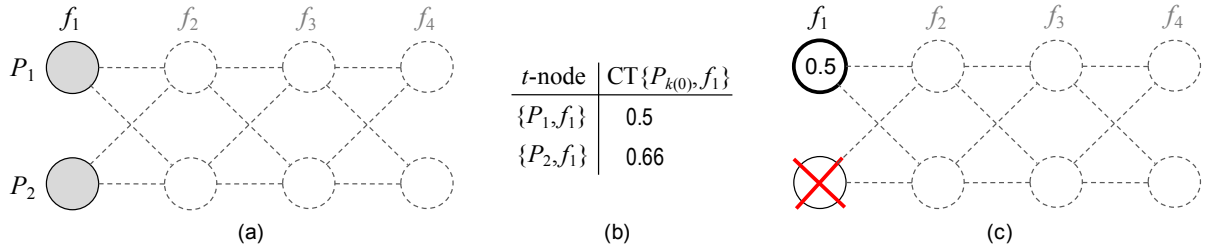


Fig. 4.22. g_w-mapping, part I: examined *t*-nodes (a), their costs (b), and the decision (c).

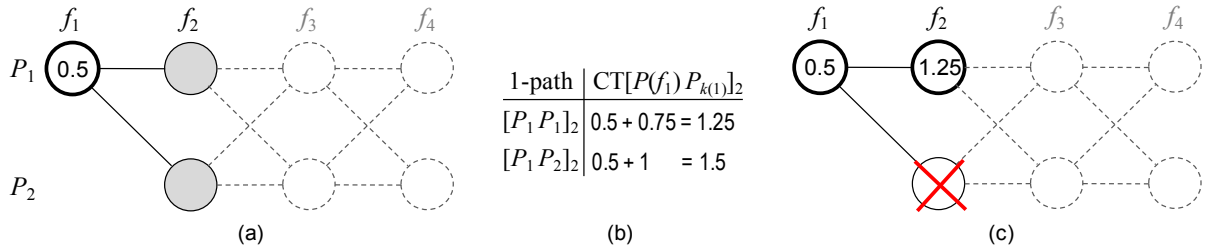


Fig. 4.23. g_1-mapping, part II: examined edges at *step* 2 (a), their costs (b), and the decision (c).

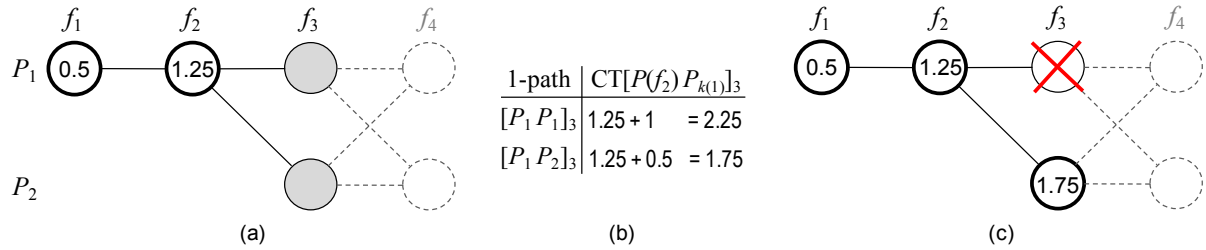


Fig. 4.24. g_1-mapping, part II: examined edges at *step* 3 (a), their costs (b), and the decision (c).

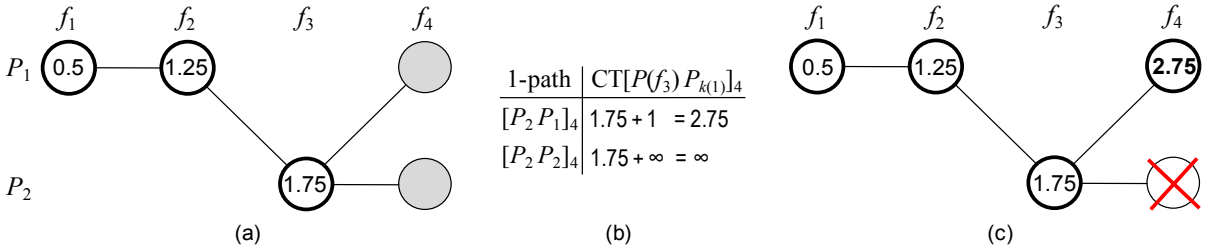


Fig. 4.25. g_1-mapping, part II: examined edges at *step* 4 (a), their costs (b), and the decision (c).

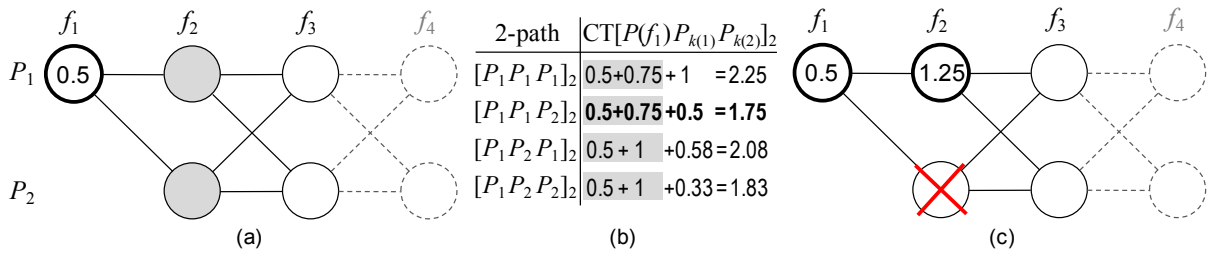


Fig. 4.26. g_2-mapping, part II: examined 2-paths at *step* 2, their costs (b), and the decision (c).

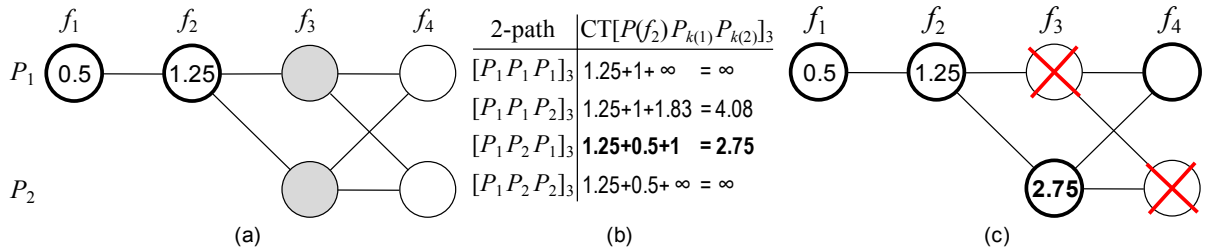


Fig. 4.27. g_2-mapping, part II: examined 2-paths at *step* 3 (a), their costs (b), and the decision (c).

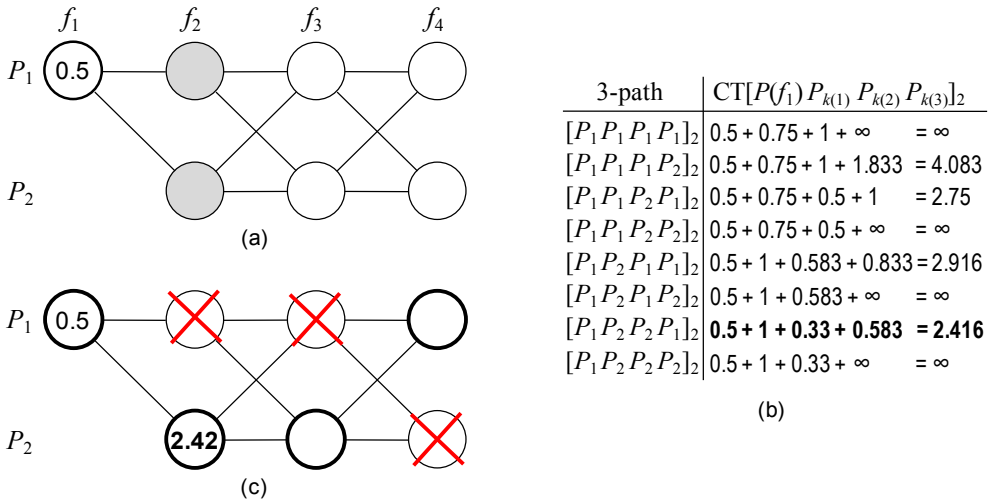


Fig. 4.28. g_3-mapping, part II: examined 3-paths at *step* 2 (a), their costs (b), and the decision (c).

Four 2-paths are examined at *step* 3 (Fig. 4.27a). Their costs (Fig. 4.27b) facilitate finding the remaining processor allocations. The minimum-cost *w*-path $[P_1\ P_2\ P_1]_3$ (Fig. 4.27b) then indicates that *t*-nodes $\{P_2, f_3\}$ and $\{P_1, f_4\}$ remain active, whereas $\{P_1, f_3\}$ and $\{P_2, f_4\}$ are discarded (Fig. 4.27c). The g_2-mapping solution—$P(f_1) = P(f_2) = P(f_4) = P_1$ and $P(f_3) = P_2$—is identical to the g_1-mapping proposal. The cost of this mapping is provided by *t*-node $\{P_2, f_3\}$, though (Fig. 4.27c).

H) g_3-mapping, Part II

The g_3-mapping evaluates all $N^w = 8$ *w*-paths between the active *t*-node at *step* 1 and a *t*-node at *step* 4 (Fig. 4.28a). Fig. 4.28b captures the costs of these 4-paths. The algorithm then selects *w*-path $[P_1\ P_2\ P_2\ P_1]_2$, which indicates the remaining processor allocations $P(f_2) = P_2$, $P(f_3) = P_2$, and $P(f_4) = P_1$ (Fig. 4.28c). The mapping cost is provided by *t*-node $\{P_2, f_2\}$ as $\mathrm{CT}\{P_2, f_2\} = 2.416$ (Fig. 4.28c).

4.3 Cost Function

The t_w-mapping selects edges and *w*-paths as a function of the premapping costs. Similarly, the g_w-mapping selects and discards *t*-nodes. These costs are obtained from some cost function. The cost function thus guides the g_w- and t_w-mapping processes. It is responsible for ensuring appropriate resource alloca-

tions under the given system constraints. The cost function, particularly, manages the computing resource availabilities and requirements and, therefore, takes advantage of the computing system modeling of Chapter 3. Table 4.2 defines the parameters and expressions that facilitate a formal cost function presentation.

4.3.1 Cost Function Template

Following the principle of templates and their instances of Chapter 3, we first propose a cost function template. We define it through edge weight $\mathrm{WT}[P_{k(l-1)}\ P_{k(l)}]_h$ as

$$\mathrm{WT}[P_{k(l-1)}\ \ P_{k(l)}]_h = \sum_{t=1}^{T} q_t \cdot cost_t^{@\{k(l),h\}} . \tag{4.8}$$

Edge $[P_{k(l-1)}\ \ P_{k(l)}]_h$ represents any edge in the t_w-mapping diagram and is for $w > 1$ part of a w-path. Ignoring q_t for the moment, each summand in (4.8) represents the accumulated cost of all necessary resource allocations of type t at t-node $\{P_{k(l)}, f_h\}$. More precisely, $cost_t^{@\{k(l),h\}}$ accounts for the resource type t specific costs of premapping SDR function f_h to processor $P_{k(l)}$.

The cost function parameter $\boldsymbol{q}$ contains the weights of the T cost terms (Table 4.2). These weights reflect the significance of each resource type: $q_t = 0$, for example, means that the computing resources of type t are either not needed or sufficiently available for the given computing resource management problem, whereas $0 < q_t \leq 1$ indicates a certain resource constraint. Without loss of generality, we normalize the sum of weights to 1; that is, $q_1 + q_2 + \ldots + q_T = 1$.

Equation (4.8) represents a template for many different cost functions. We introduce an instance of it in continuation. This cost function instance or, simply, cost function is applied in the rest of this dissertation.

4.3.2 Cost Function Instance

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. Since we consider the processing and interprocessor bandwidth capacities as the most critical SDR computing resources, we introduce a two-term cost function as an instance of (4.8). $\mathrm{WT}[P_{k(l-1)}\ P_{k(l)}]_h$ therefore defines the cost function as a superposition between the computation (processing) and the communication (data flow) costs:

$$\mathrm{WT}[P_{k(l-1)}\ \ P_{k(l)}]_h = q \cdot \mathrm{WT}_{\mathrm{COMP}}[P_{k(l-1)}\ P_{k(l)}]_h + (1-q) \cdot \mathrm{WT}_{\mathrm{COMM}}[P_{k(l-1)}\ P_{k(l)}]_h ; \tag{4.9a}$$

Table 4.2 Cost-function-specific parameters and expressions.

Parameter or expression	Range (argument range)	Description
T	$\mathbb{N}$	number of different computing resource types
t	1, 2, …, T	computing resource type
$cost_t^{@\{k(l),h\}}$	$\mathbb{R}^+$; ($k(l) \in$ 1, 2, …, N; $h \in$ 1, 2, …, M)	cost term (cost for allocating the necessary computing resources of type t at t-node $\{P_{k(l)}, f_h\}$)
q_t	[0, 1]	cost term weight
$\boldsymbol{q} = (q_1, q_2, \ldots, q_T)$		cost function parameter
$\boldsymbol{C}^{@\{k(l),\,h\}}$		remaining processing capacities at t-node $\{P_{k(l)}, f_h\}$
$C_{k(l)}^{@\{k(l),h\}}$	$\mathbb{R}^+$	remaining processing capacity of processor $P_{k(l)}$ at t-node $\{P_{k(l)}, f_h\}$
$\boldsymbol{B}^{@\{k(l),\,h\}}$		remaining bandwidths at t-node $\{P_{k(l)}, f_h\}$
$B_{l_{P(f_u)P_{k(l)}}}^{@\{k(l),h\}}$	$\mathbb{R}^+$	remaining bandwidth between processors $P(f_u)$ and $P_{k(l)}$ at t-node $\{P_{k(l)}, f_h\}$

$$\mathrm{WT}_{\mathrm{COMP}}[P_{k(l-1)}\,P_{k(l)}]_h = \begin{cases} c_h / C_{k(l)}^{@\{k(l),h\}}, & \text{if } c_h / C_{k(l)}^{@\{k(l),h\}} \le 1 \\ \infty, & \text{otherwise} \end{cases}; \tag{4.9b}$$

$$\mathrm{WT}_{\mathrm{COMM}}[P_{k(l-1)}\,P_{k(l)}]_h = \begin{cases} \sum_{u=1}^{h-1} b_{i_{uh}} / B_{I_{P(f_u)P_{k(l)}}}^{@\{k(l),h\}}, & \text{if } b_{i_{uh}} / B_{I_{P(f_u)P_{k(l)}}}^{@\{k(l),h\}} \le 1 \;\forall u \\ \infty, & \text{otherwise} \end{cases}. \tag{4.9c}$$

The cost function first computes the premapping costs at t-nodes $\{P_1, f_1\}$, $\{P_2, f_1\}$, …, and $\{P_N, f_1\}$ using the right hand side of (4.9b) and the initially available processing resources. 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; the remaining interprocessor bandwidths at each t-node of *step* 1 remain in their initial states.

Before processing edge $[P_{k(l-1)}\,P_{k(l)}]_h$ (of w-path $[P_{k(0)}\,P_{k(1)}\,\cdots\,P_{k(w)}]_i$), $\boldsymbol{C}^{@\{k(l),h\}}$ 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 $\mathrm{WT}_{\mathrm{COMP}}[P_{k(l-1)}\,P_{k(l)}]_h$ (4.9b). $\boldsymbol{B}^{@\{k(l),h\}}$, on the other hand, is dynamically updated, subtracting any required bandwidth $b_{i_{uh}}$ from the corresponding entry in $\boldsymbol{B}^{@\{k(l),h\}}$ just after adding $b_{i_{uh}}$ / $\{\cdot\}$ to $\mathrm{WT}_{\mathrm{COMM}}[P_{k(l-1)}\,P_{k(l)}]_h$ (4.9c). Fig. 4.29 illustrates this.

Each division in (4.9b) or (4.9c) is either less, equal, or greater than 1. If greater than 1, the value is mapped to infinity, which indicates infeasibility. Finite costs thus indicate feasible premappings and infinite costs infeasible solutions. This permits identifying and discarding infeasible premappings.

After a (pre)mapping decision—assuming feasibility—$\boldsymbol{C}^{@\{k(1),i\}}$ and $\boldsymbol{B}^{@\{k(1),i\}}$ remain in the states that correspond to this decision. For $i < M–w+1$, t-node $\{P_{k(1)}, f_i\}$ thus informs about the remaining computing resources of all types as a function of the (pre)mappings of SDR functions f_1, f_2, …, and f_i corresponding to the highlighted path reaching $\{P_{k(1)}, f_i\}$. For $i = M–w+1$, it contains the remaining resources after (pre)mapping all M SDR functions.

Since normalizing the sum of the cost term weights q_t (Section 4.3.1), we can write $\boldsymbol{q} = (q_1, q_2) = (q, (1–q))$ here and consider a single parameter in equation (4.9a). Parameter q may take any real value between 0 and 1, including both extremes. The cost function minimizes the data flow or communication overhead for $q = 0$ and balances the processing load for $q = 1$. We choose $q = 0.5$ by default to equally account for the limited processing and bandwidth resources.

Cost calculations	Resource updates
	$\boldsymbol{C}^{@\{k(l),h\}} = \boldsymbol{C}^{@\{k(l-1),h-1\}}$
$\mathrm{WT}_{\mathrm{COMP}}[P_{k(l-1)}P_{k(l)}]_h = \begin{cases} c_h / C_{k(l)}^{@\{k(l),h\}}, & \text{if } c_h / C_{k(l)}^{@\{k(l),h\}} \le 1 \\ \infty, & \text{otherwise} \end{cases}$	$C_{k(l)}^{@\{k(l),h\}} = C_{k(l)}^{@\{k(l),h\}} - c_h$
	$\boldsymbol{B}^{@\{k(l),h\}} = \boldsymbol{B}^{@\{k(l-1),h-1\}}$
$\mathrm{WT}_{\mathrm{COMM}}[P_{k(l-1)}P_{k(l)}]_h = \begin{cases} b_{i_{1h}} / B_{I_{P(f_1)P_{k(l)}}}^{@\{k(l),h\}}, & \text{if } b_{i_{1h}} / B_{I_{P(f_1)P_{k(l)}}}^{@\{k(l),h\}} \le 1 \\ \infty, & \text{otherwise} \end{cases}$	$B_{I_{P(f_1)P_{k(l)}}}^{@\{k(l),h\}} = B_{I_{P(f_1)P_{k(l)}}}^{@\{k(l),h\}} - b_{i_{1h}}$
$+ \begin{cases} b_{i_{2h}} / B_{I_{P(f_2)P_{k(l)}}}^{@\{k(l),h\}}, & \text{if } b_{i_{2h}} / B_{I_{P(f_2)P_{k(l)}}}^{@\{k(l),h\}} \le 1 \\ \infty, & \text{otherwise} \end{cases}$	$B_{I_{P(f_2)P_{k(l)}}}^{@\{k(l),h\}} = B_{I_{P(f_2)P_{k(l)}}}^{@\{k(l),h\}} - b_{i_{2h}}$
⋮	⋮
$+ \begin{cases} b_{i_{h-1,h}} / B_{I_{P(f_{h-1})P_{k(l)}}}^{@\{k(l),h\}}, & \text{if } b_{i_{h-1,h}} / B_{I_{P(f_{h-1})P_{k(l)}}}^{@\{k(l),h\}} \le 1 \\ \infty, & \text{otherwise} \end{cases}$	$B_{I_{P(f_{h-1})P_{k(l)}}}^{@\{k(l),h\}} = B_{I_{P(f_{h-1})P_{k(l)}}}^{@\{k(l),h\}} - b_{i_{h-1,h}}$

Fig. 4.29. Cost calculations and resource updates.

4.4 Complexity Analysis

4.4.1 General Complexity Formulation

The processing complexities of the t_w- and g_w-mapping algorithms depend 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 processing 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

$$(N + N^2 + \ \ldots \ + N^w) \cdot ccf = N \cdot \frac{N^w - 1}{N - 1} \cdot ccf \ , \tag{4.10}$$

which indicates the processing effort associated with lines 5-11 in the pseudocodes of Fig. 4.3 and Fig. 4.4, respectively.

The t_w-mapping processes N t-nodes per *step* (line 4 in Fig. 4.3) and $M–w$ steps in total (line 3). Its processing complexity then follows as

$$\text{complexity}(t_w\text{-mapping}) \approx (M - w) \cdot N^2 \cdot \frac{N^w - 1}{N - 1} \cdot ccf \ . \tag{4.11}$$

The g_w-mapping processes only one t-node per *step* and $M–w$ steps in total (line 4 in Fig. 4.4). Hence,

$$\text{complexity}(g_w\text{-mapping}) \approx (M - w) \cdot N \cdot \frac{N^w - 1}{N - 1} \cdot ccf \ . \tag{4.12}$$

These two equations do not consider the processing overheads of part I (lines 1-2 in Fig. 4.3 and lines 1-3 in Fig. 4.4) and the add-compare-store operations (lines 12-18 in Fig. 4.3 and Fig. 4.4). Equation (4.11), furthermore, ignores the t_w-mapping's postprocessing complexity (lines 19-20 in Fig. 4.3).

Part I premaps f_1 to all N processors (Sections 4.2.1A) and 4.2.2A)), whereas part III, if applicable, requires N–1 compares and following a linked list of length M for any w and cost function (Section 4.2.1C)). Equations (4.11) and (4.12) thus account for the algorithms' bulk processing overheads for typical problem sizes ($M > N$). Fig. 4.30 provides a qualitative illustration of these equations.

Since the t_w-mapping examines N times the number of w-paths that the g_w-mapping evaluates, the complexity of the t_w-mapping under some cost function is about N times the complexity of the g_w-mapping under the same cost function. That is,

$$\text{complexity}(t_w\text{-mapping}) \approx \ N \cdot \text{complexity}(g_w\text{-mapping}). \tag{4.13}$$

The complexity order of the t_w-mapping then approximately equals the complexity order of the g_{w+1}-mapping:

$$\text{complexity}(t_w\text{-mapping}) \approx \ \text{complexity}(g_{w+1}\text{-mapping}). \tag{4.14}$$

Assuming $ccf = 1$, the t_w-mapping's complexity order becomes

$$\text{complexity-order}(t_w\text{-mapping}) = O(M \cdot N^{w+1}). \tag{4.15}$$

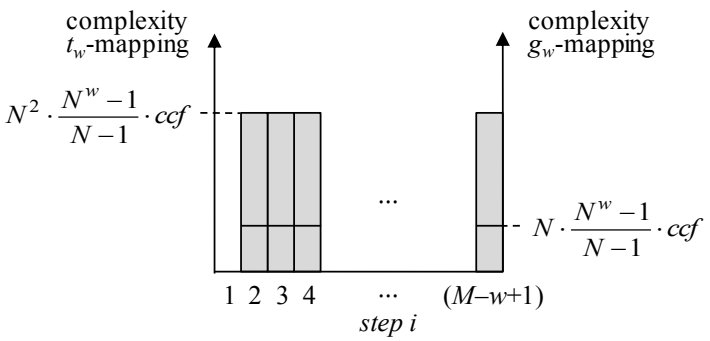


Fig. 4.30. The approximate t_w- and g_w-mapping complexities due to (4.11) and (4.12).

It indicates that the t_w-mapping algorithm is not computing efficient for large N. We therefore suggest (dynamically) dividing a large array of processors into smaller clusters of N' processors and applying the algorithm on each cluster. We have shown how processing chains and platforms can be divided into sub-chains and sub-platforms (Section 3.6.1). A hierarchical modeling (Section 3.6.2) and management may then ensure the scalability of our contribution. Without loss of generality, this dissertation assumes a small N.

4.4.2 Specific Complexity Formulation

The processing complexities associated with cost function (4.9) are not constant throughout the mapping process. Here we measure processing complexities in MAC units and present two ways for deriving the complexity of the t_w-mapping under cost function (4.9).

We assume no code optimizations and write out the number of multiplications at each forward processing step (parts I and II), for the moment ignoring the multiplications with q and $1–q$ due to (4.9a). Fig. 4.31a illustrates this.

Fig. 4.31a reveals that the number of MACs is a function of w. Fig. 4.32 illustrates this for different window sizes. A factor of type $(1 + x)$ there represents the complexity for calculating a single edge cost along a w-path. More precisely, the computation and communication cost terms (4.9b) and (4.9c) compute 1 and x MACs, respectively, for obtaining the cost increase, where x is a function of window size w, *step* i, and position l of the edge within the w-path (Fig. 4.32).

Ignoring the complexity of premapping f_1 (part I), we directly obtain

$$\text{complexity}(t_w\text{-mapping}_{(4.9)}) \approx (M-w)\cdot N^2 \cdot \sum_{l=1}^{w} N^{l-1} \cdot \frac{(M-w+2\cdot l+1)}{2} \tag{4.16}$$

from Fig. 4.31a and b. Equation (4.16) thus considers the forward processing of *steps* 2 to $M–w+1$ (lines 3-18 in Fig. 4.3). Being the most significant part of the t_w-mapping, we may consider (4.16) representative for the algorithm's complexity under cost function (4.9).

An alternative complexity derivation approach is based on writing out the complexity at step i ($i \in 2, 3, \ldots, M–w+1$).

$$\text{complexity}(t_w\text{-mapping}_{(4.9)}@\text{step } i) = [i + (i+1)\cdot N + (i+2)\cdot N^2 + \ldots + (i+w-1)\cdot N^{w-1}] \cdot N^2 \tag{4.17}$$

then directly follows from Fig. 4.33. It captures the complexity at step i of the t_w-mapping, part II. We reorganize (4.17) and obtain

$$\text{complexity}(t_w\text{-mapping}_{(4.9)}@\text{step } i) =$$
$$N^2 \cdot (1 + N + N^2 + \ldots + N^{w-1}) \cdot i + N^2 \cdot (N + 2N^2 + \ldots + (w-1) \cdot N^{w-1}). \tag{4.18}$$

$$\begin{array}{llllllll}
f_1 & & f_2 & f_3 & f_4 & \cdots & f_{M-w+1} & \\
N + N^2 \cdot \Big[& N^0 \cdot (& 2 + & 3 + & 4 + & \ldots + & (M-w+1)) & 1 \\
& + N^1 \cdot (& 3 + & 4 + & 5 + & \ldots + & (M-w+2)) & 2 \\
& \vdots & & & & & & \vdots \\
& + N^{l-1} \cdot (& (l+1) + & (l+2) + & (l+3) + & \ldots + & (M-w+l)) & l \\
& \vdots & & & & & & \vdots \\
& + N^{w-1} \cdot (& (w+1) + & (w+2) + & (w+3) + & \ldots + & M \quad) \Big] & w
\end{array}$$

(a)

Number of summands	:	$M–w$
Minimum value, 1st summand	:	$l+1$
Maximum value, $(M–w)^{\text{th}}$ summand	:	$M–w+l$

$$(l+1) + (l+1+1) + (l+1+2) + \ldots + (M-w+l) = (M-w) \cdot \frac{(M-w+l) + (l+1)}{2}$$

(b)

Fig. 4.31. Complexity derivation I for the t_w-mapping: summation (a) and factorization of MAC terms (b).

w	*step* i	MACs for processing edge $[P_{k(l-1)}\,P_{k(l)}]_{i+l-1}$ of w-path $[P_{k(0)}\,\cdots\,P_{k(w)}]_i$ $(l = 1, 2, \ldots, w)$				
		$[P_{k(0)}\,P_{k(1)}]_i$	$[P_{k(1)}\,P_{k(2)}]_{i+1}$	$[P_{k(2)}\,P_{k(3)}]_{i+2}$	$\cdots$	$[P_{k(w-1)}\,P_{k(w)}]_{i+w-1}$
1	2	$(1+1)\cdot N^2$				
	3	$+(1+2)\cdot N^2$				
	$\vdots$	$\vdots$				
	$M-2$	$+(1+(M-3))\cdot N^2$				
	$M-1$	$+(1+(M-2))\cdot N^2$				
	M	$+(1+(M-1))\cdot N^2$				
2	2	$(1+1)\cdot N^2$	$+(1+2)\cdot N^3$			
	3	$+(1+2)\cdot N^2$	$+(1+3)\cdot N^3$			
	$\vdots$	$\vdots$				
	$M-2$	$+(1+(M-3))\cdot N^2$	$+(1+(M-2))\cdot N^3$			
	$M-1$	$+(1+(M-2))\cdot N^2$	$+(1+(M-1))\cdot N^3$			
3	2	$(1+1)\cdot N^2$	$+(1+2)\cdot N^3$	$+(1+3)\cdot N^4$		
	3	$+(1+2)\cdot N^2$	$+(1+3)\cdot N^3$	$+(1+4)\cdot N^4$		
	$\vdots$	$\vdots$				
	$M-2$	$+(1+(M-3))\cdot N^2$	$+(1+(M-2))\cdot N^3$	$+(1+(M-1))\cdot N^4$		
$M-1$	2	$(1+1)\cdot N^2$	$+(1+2)\cdot N^3$	$+(1+3)\cdot N^4$	$+\ \ldots\ +(1+(M-1))\cdot N^M$	

Fig. 4.32. The t_w-mapping, part II complexities for different window sizes.

From (4.18) follows the compact formulation

$$\text{complexity}(t_w\text{-mapping}_{(4.9)}@\text{step } i) = a \cdot i + b, \tag{4.19a}$$

where

$$a = N^2 \cdot \frac{N^w - 1}{N - 1} \tag{4.19b}$$

and

$$b = N^2 \cdot \sum_{l=0}^{w-1} l \cdot N^l \tag{4.19c}$$

are constant for specific w and N ($w \in 1, 2, \ldots, M-1$; $N, M \in \mathbb{N}$).

Since (4.19) is a linear function of i (Fig. 4.34), the number of steps times the complexity at the middle step, which could be an imaginary step between two real steps, characterizes the algorithm's approximate complexity. Formally,

$$\text{complexity}(t_w\text{-mapping}_{(4.9)}) \approx (M - w) \cdot \text{complexity}(t_w\text{-mapping}_{(4.9)}@\textit{step}\ \frac{M - w + 3}{2}). \tag{4.20}$$

Note that Fig. 4.33, which led to (4.20), is merely another presentation of Fig. 4.31, from which (4.16) followed. It suffices to write out the sums in (4.16) and (4.20) and to compare terms for recognizing that both equations are equivalent.

$$\begin{aligned}
&f_1 && f_2 && f_3 && \ldots && f_i && \ldots && f_M && \\
&N + N^2 \cdot \Big[N^0 \cdot (\ 2 &&+\ 3 &&+\ \ldots + &&\ i &&+\ \ldots + &&(M-w+1)) && \Big]_1 \\
&\qquad + N^1 \cdot (\ 3 &&+\ 4 &&+\ \ldots + &&(i+1) &&+\ \ldots + &&(M-w+2)) && \Big]_2 \\
&\qquad + N^2 \cdot (\ 4 &&+\ 5 &&+\ \ldots + &&(i+2) &&+\ \ldots + &&(M-w+3)) && \Big]_3 \\
&\qquad \vdots && && && && && && && \vdots \\
&\qquad + N^{w-1} \cdot ((w+1) &&+ (w+2) &&+\ \ldots + &&(i+w-1) &&+\ \ldots + &&M\ \ ) && \Big]_w
\end{aligned}$$

Fig. 4.33. Complexity derivation II for the t_w-mapping.

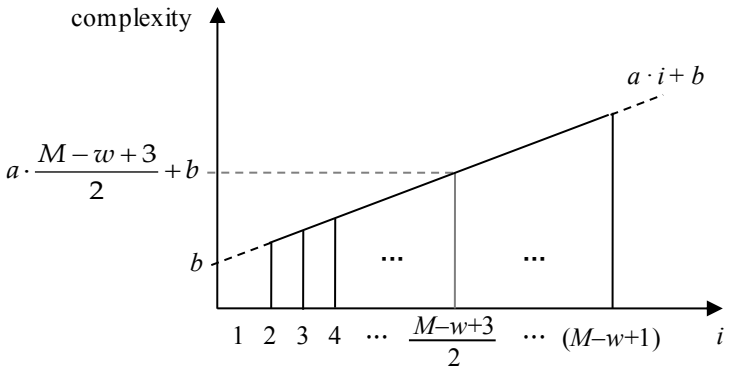


Fig. 4.34. Graphical illustration of (4.19) and (4.20).

The approximate complexities of the g_w-mapping under cost function (4.9) can be equivalently derived. The results are

$$\text{complexity}(g_w\text{-mapping}_{(4.9)}) \approx (M-w) \cdot N \cdot \sum_{l=1}^{w} N^{l-1} \cdot \frac{(M-w+2 \cdot l+1)}{2}, \tag{4.21}$$

on the one hand, and

$$\text{complexity}(g_w\text{-mapping}_{(4.9)}@\text{step } i) = a \cdot i + b \tag{4.22a}$$

with

$$a = N \cdot \frac{N^w - 1}{N-1} \tag{4.22b}$$

and

$$b = N \cdot \sum_{l=0}^{w-1} l \cdot N^l, \tag{4.22c}$$

on the other.

We now analyze the implications of cost function parameter $\boldsymbol{q} = (q, (1-q))$. If $q = (1-q) = 0.5$, the two cost terms (4.9b) and (4.9c) are equally weighted. Since weighting with $q = (1-q) = 0.5$ merely halves any partial as well as the total mapping cost without affecting any premapping decision or the final mapping outcome, we can omit the weighting. If $q \neq 0.5$, however, ignoring the multiplications with q and $1-q$ may have an effect on some premapping decisions and, thus, the mapping result. Hence, for the general case, where q may take any value in [0, 1], these multiplications cannot be neglected.

The number of additional MACs for multiplying the cost of computation (4.9b) with q and the cost of communication (4.9c) with $1-q$ depends on the implementation. Although (4.9a) suggests applying the weighting on edge basis, we consider another implementation, where the computation and the communication costs are individually calculate for the entire w-path, before weighting the cost terms while summing them up. Such an implementation would require only two extra MACs per w-path and, thus,

$$2 \cdot (M-w) \cdot N^{w+1} \tag{4.23}$$

and

$$2 \cdot (M-w) \cdot N^w \tag{4.24}$$

additional MACs for the entire t_w- and g_w-mapping procedures. Equations (4.16) and (4.21) then become

$$\text{complexity}(t_w\text{-mapping}_{(4.9)}) \approx (M-w) \cdot N^2 \cdot \left\{ 2 \cdot N^{w-1} + \sum_{l=1}^{w} N^{l-1} \cdot \frac{(M-w+2 \cdot l+1)}{2} \right\} \tag{4.25}$$

and

$$\text{complexity}(g_w\text{-mapping}_{(4.9)}) \approx (M-w)\cdot N\cdot\left\{2\cdot N^{w-1}+\sum_{l=1}^{w} N^{l-1}\cdot\frac{(M-w+2\cdot l+1)}{2}\right\}, \tag{4.26}$$

whereas

$$b = N^2\cdot\left\{2\cdot N^{w-1}+\sum_{l=0}^{w-1} l\cdot N^{l}\right\} \tag{4.27}$$

and

$$b = N\cdot\left\{2\cdot N^{w-1}+\sum_{l=0}^{w-1} l\cdot N^{l}\right\} \tag{4.28}$$

substitute (4.19c) and (4.22c).

4.4.3 General versus Specific Complexity Formulation

Here we validate the general complexity formulations (4.11) and (4.12) using the cost function specific results of Section 4.4.2. Assuming $q = 1–q = 0.5$, we equate the right-hand side of (4.11) with the right-hand side of (4.16) or, equivalently, the right-hand side of (4.12) with the right-hand side of (4.21) and obtain

$$\frac{N^w-1}{N-1}\cdot ccf = \sum_{l=1}^{w} N^{l-1}\cdot\frac{M-w+2\cdot l+1}{2}. \tag{4.29}$$

Writing out the sums on each side of (4.29) we find that, if we substitute *ccf* for $(M+w+1)/2$, (4.11) and (4.12) become upper bound formulations for complexity(t_w-mapping$_{(4.9)}$) and complexity(g_w-mapping$_{(4.9)}$), respectively. That is,

$$\text{upper-bound}(t_w\text{-mapping}_{(4.9)}) \approx (M-w)\cdot N^2\cdot\frac{N^w-1}{N-1}\cdot\frac{M+w+1}{2} \tag{4.30}$$

and

$$\text{upper-bound}(g_w\text{-mapping}_{(4.9)}) \approx (M-w)\cdot N\cdot\frac{N^w-1}{N-1}\cdot\frac{M+w+1}{2}. \tag{4.31}$$

For $w = 1$, these upper bound formulations coincide with the approximate complexity equations (4.16) and (4.21). For $w > 1$, the smaller N and M the higher the deviation of an upper bound from the corresponding reference value: The deviation is up to a 9 % for $N = 2$ and $M = 15$. For $N \geq 3$ and $M \geq 15$, the maximum deviation of 4.8 % is obtained for $M = 15$ and $w = 5$. The practical values $N = 3$ and $M = 24$ (Section 3.5), lead to a maximum deviation of 3.3 %. Hence, (4.30) and (4.31) are relatively good approximations of the t_w- and g_w-mapping complexities under cost function (4.9).

4.5 SDR Computing Resource Management Example

Our computing resource management approach relies on an appropriate computing system modeling, which provides the information about the available and required computing resources. The computing resource management module manages these resources and dynamically updates their states after each resource allocations or deallocation (Fig. 4.35).

Fig. 4.36 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 UMTS task graph of Fig. 3.7a to SDR platform IV (Fig. 3.5a) using cost function (4.9). Hence, P_1, P_2, and P_3 correspond to $(P_1)^{IV}$, $(P_2)^{IV}$, and $(P_3)^{IV}$ and $f_1, f_2, \ldots, f_{24}$ to $(f_1)^{UMTS}$, $(f_2)^{UMTS}$, …, $(f_{24})^{UMTS}$ here. The processing and data flow requirements are then $\boldsymbol{c}^{\mathbf{UMTS}}$ and $\boldsymbol{b}^{\mathbf{UMTS}}$ (Fig. 3.7b) and the initial processing and bandwidth resources $\boldsymbol{C}^{\mathbf{IV}}$ and $\boldsymbol{B}^{\mathbf{IV}}$ (Fig. 3.5b). 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

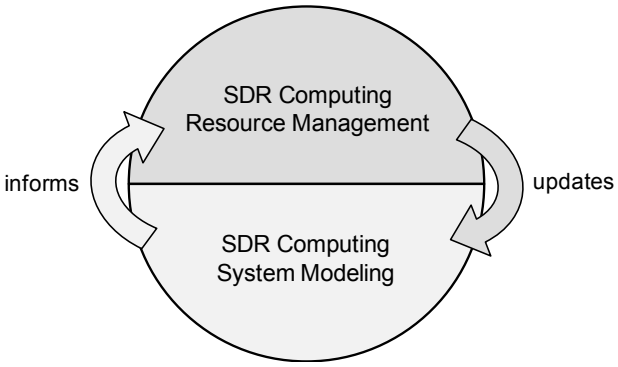


Fig. 4.35. Interactions between management and modeling.

- $(c_1)^{\mathrm{UMTS}} / (C_1)^{\mathrm{IV}} = 0.0765 / 8.823 = 8.67 \cdot 10^{-3}$,
- $(c_1)^{\mathrm{UMTS}} / (C_2)^{\mathrm{IV}} = 0.0765 / 5.882 = 13 \cdot 10^{-3}$, and
- $(c_1)^{\mathrm{UMTS}} / (C_3)^{\mathrm{IV}} = 0.0765 / 2.941 = 26.01 \cdot 10^{-3}$

and stored at t-nodes $\{P_1, f_1\}$, $\{P_2, f_1\}$, and $\{P_3, f_1\}$, respectively. These t-nodes also provide the remaining processing powers $\boldsymbol{C}^{\mathbf{IV@\{k(l),\,1\}}}$ and bandwidths $\boldsymbol{B}^{\mathbf{IV@\{k(l),\,1\}}}$:

- $\boldsymbol{C}^{\mathbf{IV@\{1,\,1\}}} = ((C_1)^{\mathrm{IV}} - (c_1)^{\mathrm{UMTS}}, (C_2)^{\mathrm{IV}}, (C_3)^{\mathrm{IV}}) = (8.746, 5.882, 2.941)$ MOPTS,
- $\boldsymbol{C}^{\mathbf{IV@\{2,\,1\}}} = ((C_1)^{\mathrm{IV}}, (C_2)^{\mathrm{IV}} - (c_1)^{\mathrm{UMTS}}, (C_3)^{\mathrm{IV}}) = (8.823, 5.805, 2.941)$ MOPTS,
- $\boldsymbol{C}^{\mathbf{IV@\{3,\,1\}}} = ((C_1)^{\mathrm{IV}}, (C_2)^{\mathrm{IV}}, (C_3)^{\mathrm{IV}} - (c_1)^{\mathrm{UMTS}}) = (8.823, 5.882, 2.864)$ MOPTS, and
- $\boldsymbol{B}^{\mathbf{IV@\{1,\,1\}}} = \boldsymbol{B}^{\mathbf{IV@\{2,\,1\}}} = \boldsymbol{B}^{\mathbf{IV@\{3,\,1\}}} = \boldsymbol{B}^{\mathbf{IV}}$.

As explained in Section 4.2.1B), the t_w-mapping evaluates N^w w-paths. The tables in Fig. 4.36 contain the accumulated w-path costs $\mathrm{CT}[\cdot]_2$ due to Fig. 4.3, line 12. Both algorithm instances highlight edge $[P_2\ P_2]_2$, which results from the minimum-cost w-paths $[P_2\ P_2]_2$ and $[P_2\ P_2\ P_2]_2$, respectively. The accumulated cost,

- $\mathrm{CT}\{P_2, f_2\} = \mathrm{CT}\{P_2, f_1\} + \mathrm{CT}[P_2\ P_2]_2 = 0.013 + 0.0499 = 0.0629$,

and the remaining resources,

- $\boldsymbol{C}^{\mathbf{IV@\{2,\,2\}}} = ((C_1)^{\mathrm{IV@\{2,\,1\}}}, (C_2)^{\mathrm{IV@\{2,\,1\}}} - (c_2)^{\mathrm{UMTS}}, (C_3)^{\mathrm{IV@\{2,\,1\}}}) = (8.823, 5.516, 2.941)$ MOPTS and
- $\boldsymbol{B}^{\mathbf{IV@\{2,\,2\}}} = \boldsymbol{B}^{\mathbf{IV@\{2,\,1\}}} = \boldsymbol{B}^{\mathbf{IV}}$,

are then stored at t-node $\{P_2, f_2\}$.

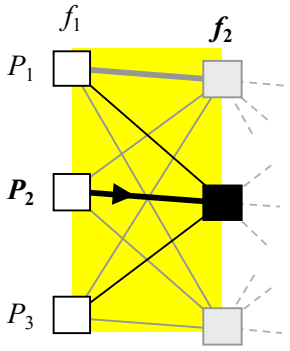
The remaining t-nodes up to those at *step* M–w+1 are correspondingly processed. This finishes part II of the algorithm. Due to Section 4.2.1C), we obtain the final t_w-mapping by traversing the t-nodes along the highlighted edges starting at the minimum-cost t-node at *step* M–w+1. The minimum-cost t-node is obtained from cost comparisons: For the t_1-mapping example, the accumulated costs at t-nodes $\{P_1, f_{24}\}$, $\{P_2, f_{24}\}$, and $\{P_3, f_{24}\}$ are 4.3902, 4.3896, and 4.3891. Hence, t-node $\{P_3, f_{24}\}$ is the starting point of the backtracking process, which Fig. 4.37a illustrates. Eventually, f_1, f_2, …, f_5, f_7, f_9, f_{10}, f_{11}, f_{13}, and f_{15} are mapped to P_1, f_6 and f_8 to P_2, and f_{12}, f_{14}, f_{16}, f_{17}, …, f_{24} to P_3. The cost of this mapping is $\mathrm{CT}\{P_3, f_{24}\} = 4.3891$. The remaining computing resources are also provided by t-node $\{P_3, f_{24}\}$. These are

- $\boldsymbol{C}^{\mathbf{IV@\{3,\,24\}}} = (0.3792, 1.1765, 2.4166)$ MOPTS and
- $\boldsymbol{B}^{\mathbf{IV@\{3,\,24\}}} = \begin{pmatrix} \infty & 0.3041 & 0.2896 \\ 0.8101 & \infty & 0.5882 \\ 0.2941 & 0.5882 & \infty \end{pmatrix}$ MBPTS.

Fig. 4.37b illustrates part III of the t_2-mapping example. The minimum cost t-node at *step* M–1 is $\{P_3, f_{23}\}$, being 4.3706, 4.3633, and 4.3601 the costs stored at t-nodes $\{P_1, f_{23}\}$, $\{P_2, f_{23}\}$, and $\{P_3, f_{23}\}$. After finishing the postprocessing phase, P_1 is assigned to f_1, f_2, …, f_6, f_8, f_{10}, and f_{11}, P_2 to f_7 and f_9, and P_3 to f_{12}, f_{13}, …, f_{24}. The computing resources remain in the following states:

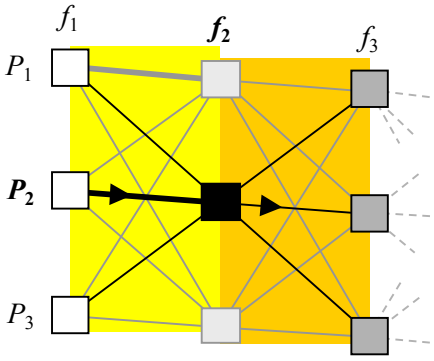
- $\boldsymbol{C}^{\mathbf{IV@\{3,\,23\}}} = (0.5439, 1.1765, 2.2519)$ MOPTS and

- $\boldsymbol{B}^{\mathrm{IV}@\{3,\,23\}} = \begin{pmatrix} \infty & 0.3041 & 0.2218 \\ 0.8101 & \infty & 0.5882 \\ 0.2941 & 0.5882 & \infty \end{pmatrix}$ MBPTS.



1-path	$\mathrm{CT}[P_{k(0)}\,P_2]_2$
$[P_1\ P_2]_2$	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	$\mathrm{CT}[P_{k(0)}\,P_2\,P_{k(2)}]_2$
$[P_1\,P_2\,P_1]_2$	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. 4.36. Part II of the t_w-mapping example: processing at t-node $\{P_2, f_2\}$ for $w = 1$ (a) and $w = 2$ (b).

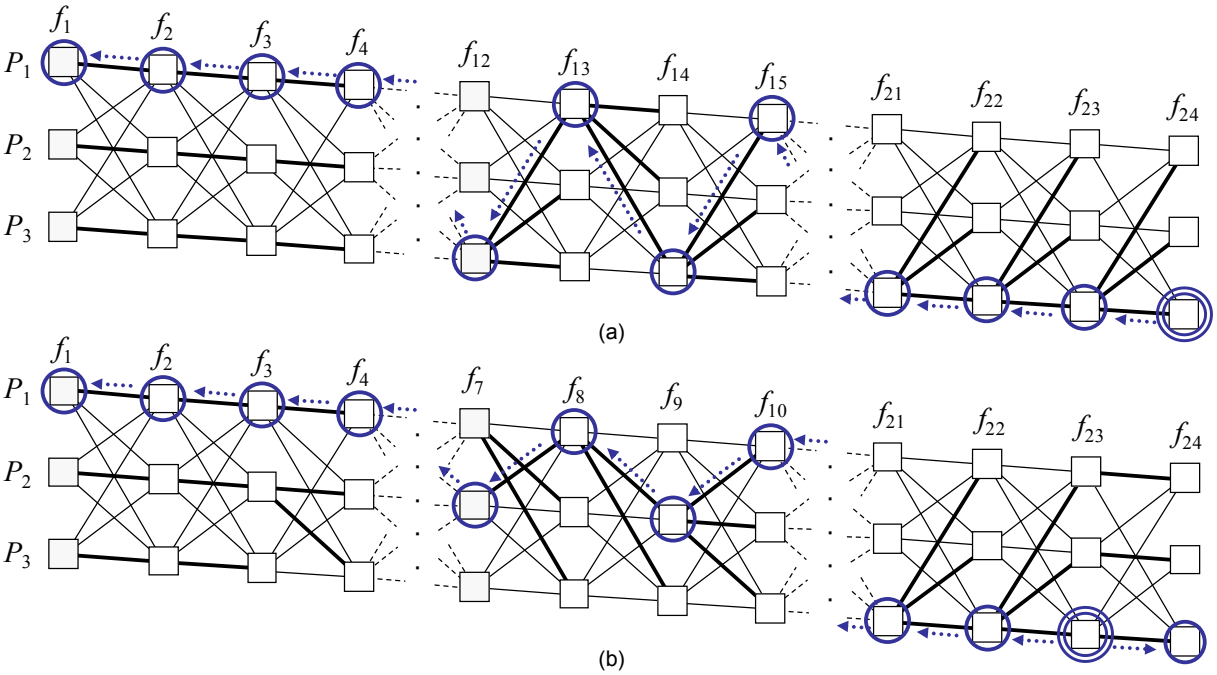


Fig. 4.37. Part III of the t_w-mapping example for $w = 1$ (a) and $w = 2$ (b).

Table 4.3 t_w-mapping solutions and costs for different window sizes.

w	Mapping solution			Cost
	P_1	P_2	P_3	
1	$f_1, \ldots, f_5, f_7, f_9, f_{10}, f_{11}, f_{13}, f_{15}$	f_6, f_8	$f_{12}, f_{14}, f_{16}, \ldots, f_{24}$	4.3891
2, …, 5	$f_1, \ldots, f_6, f_8, f_{10}, f_{11}$	f_7, f_9	$f_{12}, \ldots, f_{24}$	4.3601

The cost of the t_2-mapping solution, $CT\{P_3, f_{23}\} = 4.3601$, is slightly lower than that of the t_1-mapping result (Table 4.3). The t_3-, t_4-, and t_5-mapping outcomes are identical to the t_2-mapping proposal. These mappings, moreover, coincide with an optimal solution for the given problem and cost function.

4.6 Summary

This chapter has presented the second part of our SDR computing resource management framework. We have introduced a general-purpose mapping algorithm, the t_w-mapping, and a multi-objective cost function, which guides the mapping processes and implements the mapping policy. We have also formally described an extended greedy algorithm, the g_w-mapping, which will be applied for the performance evaluations of Chapter 5. The theoretical complexity analyzes have shown that the processing complexity of the t_w-mapping is N times the processing complexity of the g_w-mapping, where N stands for the number of processors. We have furthermore shown that the complexity order of the t_w-mapping matches that of the g_{w+1}-mapping. These theoretical results are the basis for the analyses that follow.

5

SDR Scenarios and Simulations

5.1 Introduction

We have presented an SDR computing resource management framework that consists of an SDR computing system modeling (Chapter 3) and a computing resource management (Chapter 4). The entire framework needs to be evaluated in realistic SDR scenarios. This chapter therefore simulates possible SDR computing resource management problems and evaluates the framework's dynamic reconfiguration capabilities. It analyzes the mapping results in terms of performance versus complexity for two simulations scenarios (Sections 5.2 and 5.3).

We already mentioned that it is not feasible to adapt previously introduced algorithms to the SDR computing resource management context and to evaluate their performance within our framework. Because using these algorithms is impractical [49], we consider the t_w-mapping─our algorithm proposal─and the g_w-mapping─the baseline reference algorithm. The simplicity of the g_w-mapping makes it applicable to realistic scenarios, which are often very complex. Its parametric design, on the other hand, permits a flexible adjustment to the particular problem and facilitates the comparison with the t_w-mapping. Each baseline result is complemented with the optimal solution, obtained from an exhaustive search.

5.2 UMTS Task Graph

5.2.1 Scenario – Simulation Setup

A future SDR platform will be subject to dynamic reconfigurations 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

$$(c\text{-}load,\ b\text{-}load) = \left(\frac{(c_T)^d}{sf_C \cdot (C_T)^D},\ \frac{(b_T)^d}{sf_B \cdot (B_T)^D} \right), \tag{5.1}$$

where *c-load* ∈ 0.2, 0.35, …, 0.95, and *b-load* ∈ 0.75, 1.25, …, 3. *c-load* specifies the relation between the total processing requirement of SDR application d and the total processing power of SDR platform D after scaling. *b-load* relates the application's total bandwidth demand $(b_T)^d$ to a platform's bandwidth capacity $sf_B \cdot (B_T)^D$.

The SDR application corresponds to the UMTS downlink receiver of Fig. 3.7. The scenario considers the four SDR platforms I-IV of Fig. 3.3; Fig. 3.5 illustrates the system models of platform IV.

We assume that the SDR platforms contain the analog parts and enough computing resources for executing the corresponding uplink transmitter and the other processing layers of the UMTS transceiver. Hence, we consider the mapping of the processing chain of Fig. 3.7a as the critical computing resource management task here. We apply cost function (4.9) with $q = (1–q) = 0.5$ and the t_w- and g_w-mapping algorithms with different window sizes.

5.2.2 Simulation Results

Fig. 5.1 shows the g_w- and Fig. 5.2 the t_w-mapping results. A square in each subfigure represents a particular (*c-load*, *b-load*) tuple, which specifies the mapping problem. Its shading indicates whether the corresponding g_w- 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 an 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 is feasible. We call this an *impossible mapping situation*.

From Fig. 5.1 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 considered mapping problems.

We find that the t_w-mapping ($w \in 1, 2, ..., 5$) is more robust against *c-load* and *b-load* variations than the g_w-mapping. The most critical difference between the algorithms is observed for SDR platform II: While the t_1-mapping achieves feasible results for all possible mapping situations, the g_w-mapping fails in nine cases for $w \leq 3$, in five for $w = 4$, and in four for $w = 5$ (black squares in Fig. 5.1b, f, j, n, and r). Also, the number of optimal t_w-mapping results is considerably higher than the number of optimal g_w-mappings (white squares in Fig. 5.1 and Fig. 5.2).

In Section 4.4.1 we found that the t_w- and g_{w+1}-mapping processing complexities. (This is further analyzed in Section 5.2.3.) We therefore compare the t_w-mapping with the g_{w+1}-mapping results and observe that the t_w-mappings are closer to the optimal solutions for most states of any SDR platform. For example, 30 out of 43 possible mappings at SDR platform IV are optimal when applying the t_1-mapping algorithm, whereas only 17 are optimal with the g_2-mapping. The t_2-mapping is capable of optimally solving all possible mapping situations at platform III; the g_w-mapping achieves this for $w = 5$. The g_5-mapping is, however, two orders of magnitude more complex than the t_2-mapping due to (4.14) and (4.15). The t_1-mapping, moreover, feasibly solves all but one solvable mapping problem, whereas the g_3-mapping, which is N times more complex then the t_1-mapping, fails in ten cases.

Fig. 5.1 and Fig. 5.2 also indicate that a higher w does not necessarily lead to a better result. The t_{M-1}-mapping implementation guarantees finding an optimal mapping, whereas the g_w-mapping for any w and the t_w-mapping for $w < M–1$ are suboptimal for the general mapping problem. A higher window size promises a better mapping solution on average, as the following simulations will show, rather than ensuring a better result for a particular mapping problem.

Neither the t_4- nor the t_5-mapping achieves optimal results for all resource conditions of architectures II and IV. The t_2-mapping, on the other hand, achieves optimal results for (almost) all conditions of architectures I and III. The significant complexity difference between the t_2-mapping and the optimal t_{23}-mapping is essential in dynamic reconfiguration scenarios. The frequent initializations and terminations of sessions we currently find in base stations, for example, require a computing efficient mapping approach that provides the necessary performance. A feasible solution may be sufficient here, whereas an optimal solution may 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.

5.2.3 Practical versus Theoretical Complexities

We consider the algorithm execution times as the practical complexity indicators. Equations (4.16) and (4.21) specify the corresponding theoretical complexities. We have implemented the g_w- and t_w-mapping algorithms in C. In order to make code optimizations less significant, we measure the execution times for

(*c-load*, *b-load*) = (0.2, 0.75) and platform I. This scenario permits $1.08 \cdot 10^{11}$ feasible mappings out of $3^{24} = 2.82 \cdot 10^{11}$ different mapping solutions.

Table 5.1 contains the approximate execution times (ETs) of the entire t_1-, t_2-, t_3-, t_4-, and t_5-mapping processes (parts I, II, and III) on a 3 GHz GPP. Table 5.2 shows the corresponding execution times of the g_w-mapping implementation. The tables also show the theoretical complexities due to (4.16) and (4.21), respectively. The last two columns provide the execution times and MACs scaled by the execution time and MAC for $w = 1$.

The scaled ETs and the scaled MACs of Table 5.1 do not match very well. One reason could be the overhead associated with the relatively large number of array declarations and conditional updates, which permit an embedded implementation of the t_w-mapping for different window sizes ($w \in 1, 2, ..., 5$) while increasing the execution time of the t_1-mapping. Another reason would be the code optimizations, which are the more effective the larger w: Once identifying infeasibility, the algorithm immediately stops computing the costs of the associated w-paths.

The practical and theoretical complexity approximations for the g_w-mapping, on the other hand, match very well (Table 5.2). Despite the same implementation principle, the g_w-mapping needs fewer matrix declarations and updates because of the straightforward mapping procedure. The decreasing ratio between the scaled ETs and the scaled MACs for increasing w is in line with the above mentioned code optimizations.

Table 5.1 Execution times and theoretical complexities of the t_w-mapping.

w	Execution time (ET) [μs]	MACs (4.16)	Scaled ET	Scaled MACs
1	46.9	2 691	1.0	1.0
2	139.0	10 494	3.0	3.9
3	393.8	33 453	8.4	12.4
4	1 114.1	101 160	23.8	37.6
5	3 196.8	300 447	68.2	111.6

Table 5.2 Execution times and theoretical complexities of the g_w-mapping.

w	Execution times (ETs) [µs]	MACs (4.21)	Scaled ET	Scaled MACs
1	10.9	897	1.0	1.0
2	43.7	3 498	4.0	3.9
3	132.9	11 151	12.2	12.4
4	384.3	33 720	35.3	37.6
5	1 109.4	100 149	101.8	111.6

Table 5.3 t_w- versus g_{w+1}-mapping, execution times.

w	t_w-mapping (*) [μs]	g_{w+1}-mapping [μs]	Absolute and relative differences relative to (*)	
1	46.9	43.7	-3.2	-6.8 %
2	139.0	132.9	-6.1	-4.4 %
3	393.8	384.3	-9.5	-2.4 %
4	1 114.1	1 109.4	-4.7	-0.4 %

Table 5.4 t_w- versus g_{w+1}-mapping, theoretical complexities.

w	MACs t_w-mapping (*)	MACs g_{w+1}-mapping	Absolute and relative differences relative to (*)	
1	2 691	3 498	+807	+30.0 %
2	10 494	11 151	+657	+6.3 %
3	33 453	33 720	+267	+0.8 %
4	101 160	100 149	-1 011	-1.0 %
5	300 447	295 002	-5 445	-1.8 %
7	2 592 585	2 519 232	-73 353	-2.8 %
9	21 921 435	21 080 346	-841 089	-3.8 %

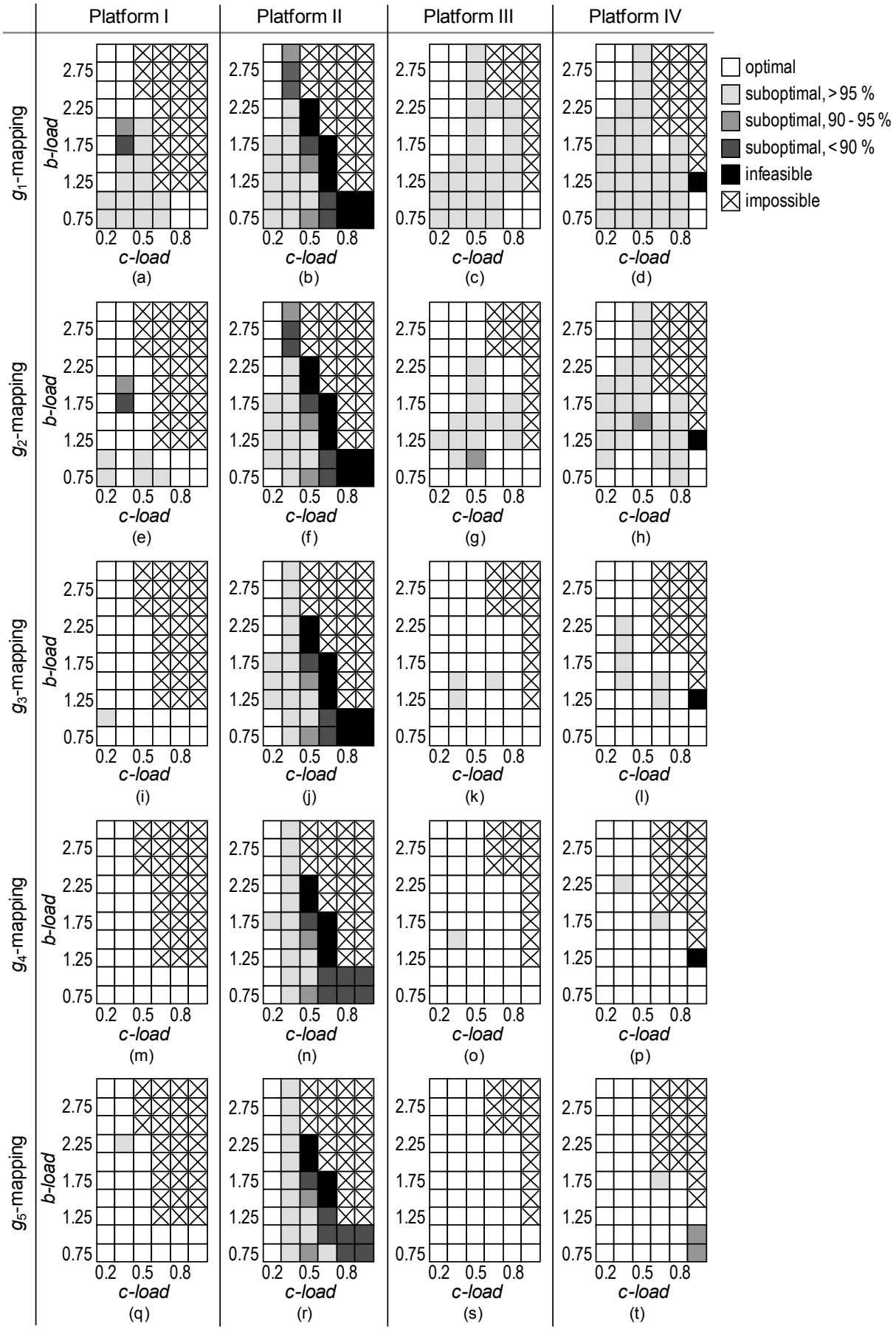


Fig. 5.1. g_1- (a-d), g_2- (e-h), g_3- (i-l), g_4- (m-p), and g_5-mapping results (q-t) for SDR platforms I-IV.

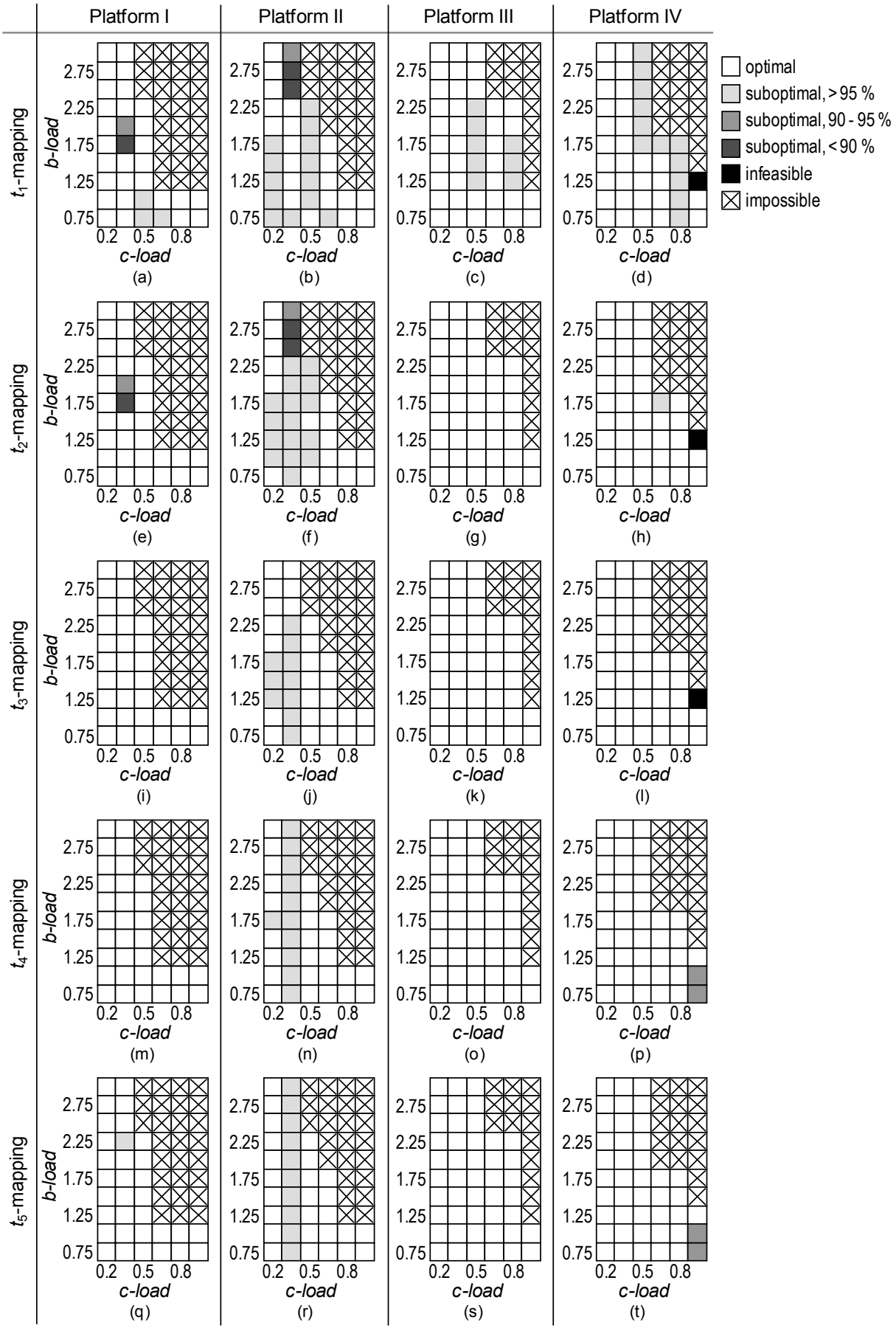


Fig. 5.2. t_1- (a-d), t_2- (e-h), t_3- (i-l), t_4- (m-p), and t_5-mapping results (q-t) for SDR platforms I-IV.

We finally compare the t_w-mapping and g_{w+1}-mapping complexities based on the figures of Table 5.1 and Table 5.2. Table 5.3 captures the execution times. We observe relatively good matches, especially for higher w: The g_2-mapping computes the mapping of the UMTS task graph to SDR platform I in 3.2 µs or 6.8 % less time than the t_1-mapping, whereas the g_5-mapping finishes 4.7 µs or only 0.4 % earlier than the t_4-mapping. The more complex execution control of the t_w-mapping implementation explains the faster execution time of the g_{w+1}-mapping process for any $w \in 1, 2, 3, 4$.

Table 5.4 provides the MACs of the t_w- and g_{w+1}-mapping algorithms. We observe a relatively large difference for $w = 1$: The g_2-mapping executes 807 or 30 % more MACs than the t_1-mapping. The absolute and relative differences are much lower for $w = 2$ and practically negligible for $w = 3$. The deviations for $w = 1$ and 2 further support the t_w-mapping, which achieves better mapping results than the g_{w+1}-mapping (Section 5.2.2) at lower computing costs (Table 5.4).

In the theoretical analysis of Section 4.4 we found that the processing complexities of the t_w- and g_{w+1}-mapping approximately match. The practical SDR scenario and different complexity indicators confirm this result here.

5.3 Random Task Graphs

5.3.1 Scenario – Simulation Setup

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 SDR-MTs; Fig. 3.3 shows their computing architectures and resources. Each terminal is reconfigured one million times. A reconfiguration of an SDR platform consists of de-mapping the old SDR application and mapping the new (total reconfiguration).

We randomly generate 1 000 000 DAGs based on the following parameters:

- the number of nodes per DAG is $M(d) = 18$,
- the data flow connection probability is $con^d = 0.15$,
- the processing demands $(c_u)^d$ ($u \in 1, 2, \ldots, M(d)$) are uniformly distributed in {1, 2, ..., 2500} MOPS, and
- the bandwidth demands $(l_{uv})^d$ ($u, v \in 1, 2, \ldots, M(d)$) of existing data flow connections are uniformly distributed in {1, 2, ..., 500} Mbps.

These DAGs should be understood as different SDR applications that represent different RATs or RAT variations. In the mean they require 75 % of a platform's total processing resources. The connectivity parameter con^d here indicates the probability of connecting, or drawing an arc between, any two nodes in the DAG while following the logical numbering rule. That is, the probability of connecting $(f_u)^d$ to $(f_v)^d$ is 0.15, if $v > u$ and 0, otherwise. We allow disconnected graphs (a graph consisting of two or more connected subgraphs or components [104]), which model parallel function chains, but connect any isolated node $(f_u)^d$ to its next neighbor. A two-component DAG could then, for example, represent an SDR transceiver model with independent function chains for the transmit and receive paths. Irrespective of the particular DAG, we specify the time slot duration t_{TS} as $0.5 \cdot 10^{-3}$ SPTS. The latency will then be $M(d) \cdot t_{TS} =$ 9 ms (3.15) at most.

5.3.2 Simulation Results

The performance metric is the percentage of infeasible mappings. Associated with lost or interrupted user sessions, the objective is minimizing the number of infeasible reconfigurations. We again apply cost function (4.9) with $q = (1–q) = 0.5$ and consider the window sizes 1 to 5.

Fig. 5.3 illustrates the percentage of infeasible g_w- and t_w-mappings. These results exclude the 6608 DAGs (0.7 %) that require more processing resources than available. Additional simulations revealed that practically all considered DAGs, more than 99.99 %, can be feasibly mapped to any of the four SDR platforms.

We observe that the number of infeasible g_w- or t_w-mappings decreases with increasing window sizes. Fig. 5.4 shows that this decrease is exponential. (The solid lines in Fig. 5.4 are exponential tendencies.) As w increases from 1 to 5, the number of infeasible mappings decreases more significantly for SDR platform III or IV (Fig. 5.4b) than for platform I or II (Fig. 5.4a). Comparing the t_w- with the g_w-mapping results we find that the number of unfeasibly t_w-mapped DAGs is about half the number of unfeasibly g_w-mapped DAGs for any platform and window size (Table 5.5).

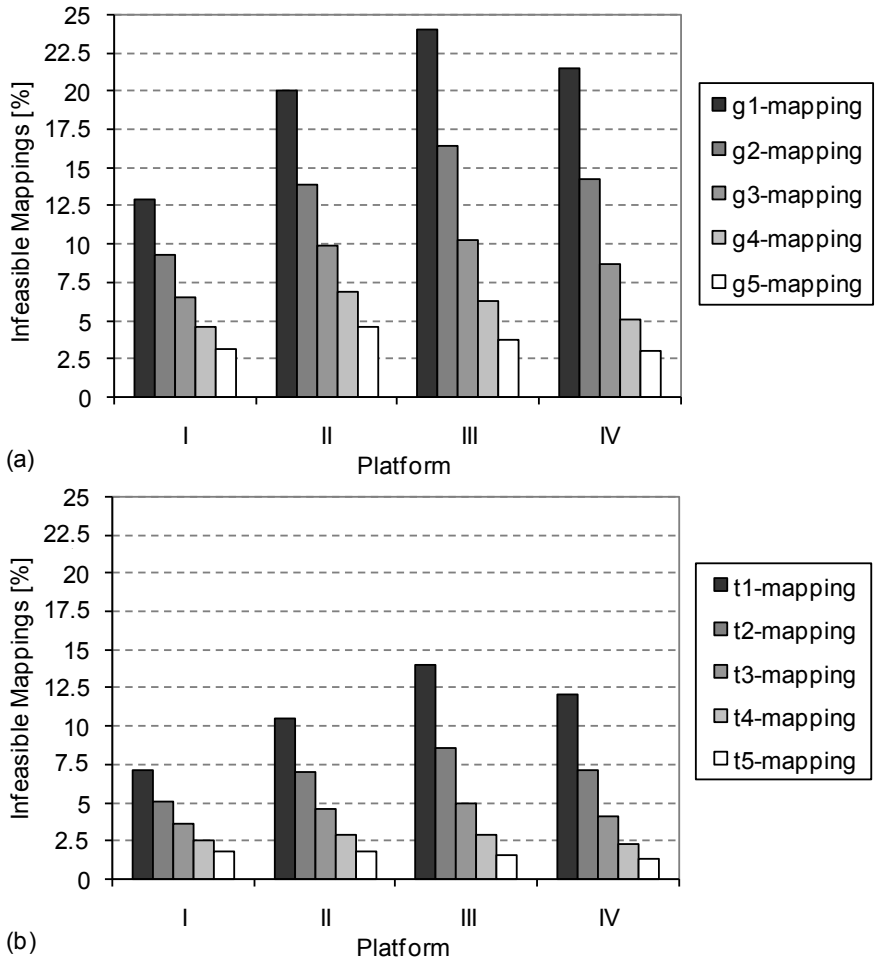


Fig. 5.3. Percentage of infeasible g_w- (a) and t_w-mappings (b) for SDR platforms I-IV.

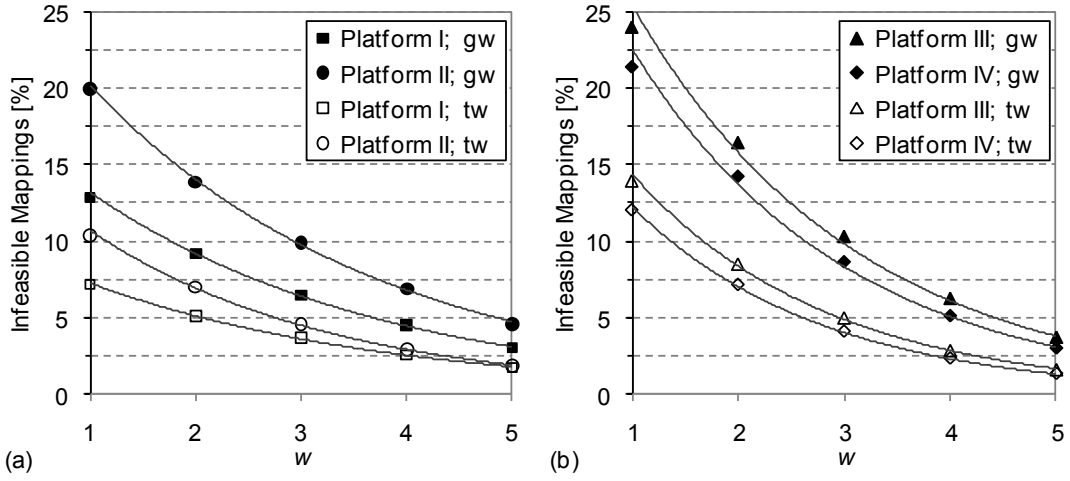


Fig. 5.4. Percentage of infeasible g_w- and t_w-mappings for SDR platforms I and II (a) and III and IV (b).

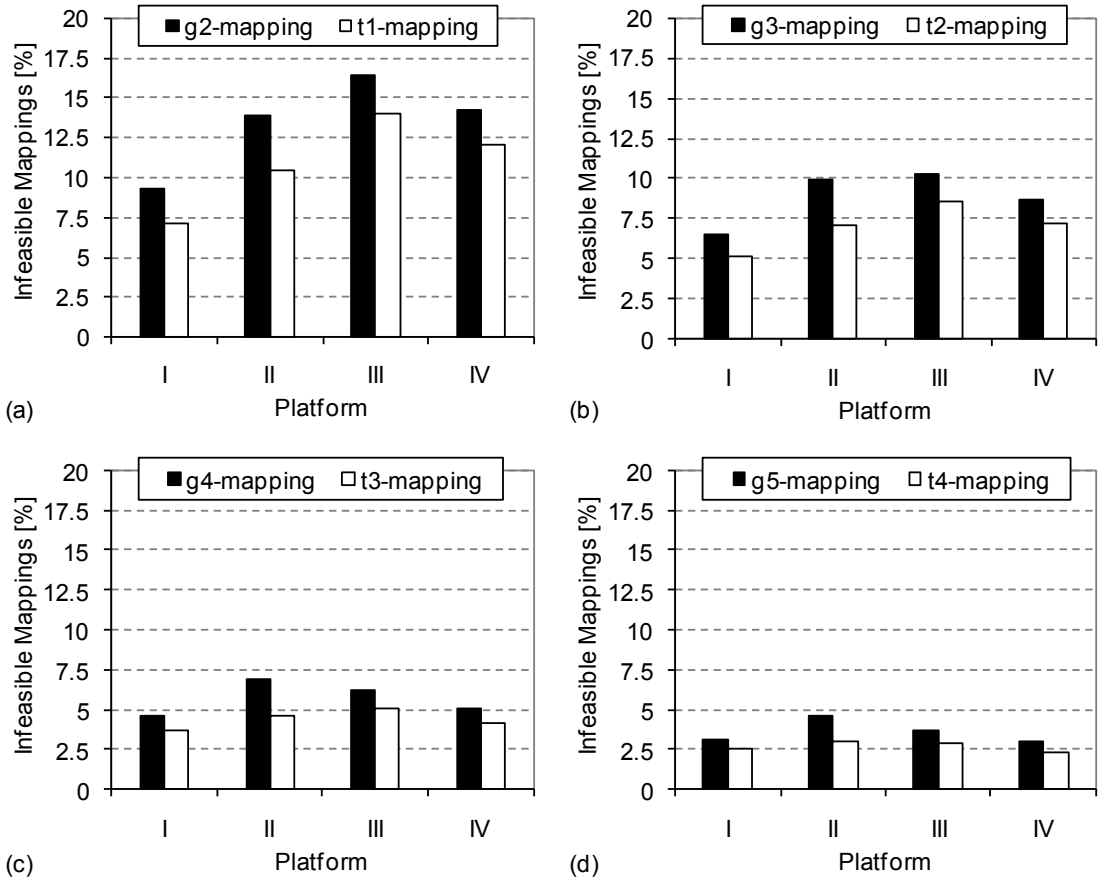


Fig. 5.5. Percentage of infeasible t_w- versus g_{w+1}-mappings for $w = 1$ (a), $w = 2$ (b), $w = 3$ (c), and $w = 4$ (d).

Because of the equivalency between the t_w- and g_{w+1}-mapping complexities (Sections 4.4 and 5.2.3), we compare the t_w- with the g_{w+1}-mapping results. Fig. 5.5a-d therefore illustrate the infeasible t_w-mappings side by side with the infeasible g_{w+1}-mappings for $w = 1$, 2, 3, and 4. These results show that the t_w-mapping feasibly maps more random DAGs for any window size and SDR platform. Its absolute gain ranges from 5222 ($w = 4$, platform I) to 34 182 ($w = 1$, platform II) additionally mapped DAGs. The relative gain over the infeasible g_{w+1}-mappings varies from 15.2 % ($w = 1$, platform III) to 36 % ($w = 4$, platform II) as Table 5.6 indicates.

We finally analyze the necessary window sizes for different performance limits. If, for example, we set the limit to 7.5 % infeasible mappings, we observe that the t_1-mapping is appropriate for SDR platform I, the t_2-mapping for platforms II and IV, whereas $w = 3$ is necessary for platform III (Fig. 5.3b). If the limit is 5 %, the t_3-mapping is appropriate for all platforms, whereas a limit of 2.5 % requires the t_4-mapping for platform IV and the t_5-mapping otherwise.

Table 5.5 Percentage of infeasible t_w-mappings with respect to the infeasible g_w-mappings.

Platform \ w	1	2	3	4	5
I	55.3	55.2	55.9	56.5	56.6
II	52.3	50.6	46.2	42.5	39.7
III	58.1	51.8	48.6	45.9	43.8
IV	56.2	50.2	47.4	45.6	44.5

Table 5.6 Percentage of infeasible t_w-mappings with respect to the infeasible g_{w+1}-mappings.

Platform \ w	1	2	3	4
I	77.1	78.1	80.5	83.0
II	75.3	70.7	66.2	64.0
III	84.8	82.7	80.3	77.7
IV	84.5	82.8	80.2	78.1

The g_w-mapping can meet the 7.5 % performance limit with practical window sizes: $w = 3$ for platform I and $w = 4$ for platforms II-IV (Fig. 5.3a). Larger windows are, however, necessary for the 5 % and 2.5 % boundaries. These results qualify the algorithms' performances. Despite the more general simulation setup, we can make similar conclusions as for the first SDR scenario.

5.4 Summary

In this chapter we have applied the SDR computing resource management framework of Chapters 3 and 4 for solving different SDR reconfiguration scenarios. The simulation results have demonstrated that our contribution is suitable for solving different computing resource management problems. Optimal results are often achieved for window sizes as small as 1, 2, or 3.

We have compared the t_w-mapping with the g_w-mapping outcomes and have observed that the t_w-mapping algorithm achieves considerably better mappings. We have confirmed the theoretical result of Section 4.4, where we found that the processing efforts associated with the t_w-mapping and g_{w+1}-mapping approximately correspond. Nevertheless, the t_w-mapping has outperformed the g_{w+1}-mapping for different performance criteria.

The simulation results have also indicated a dependency between the SDR platform and the application mapping. We have, particularly, observed that the choice of an appropriate window size is a function of the SDR platform. Since the processing complexity of the t_w-mapping algorithm is proportional to N^{w+1} (4.15), w should not be overdimensioned, but, rather, just as high as necessary. Chapter 6 analyzes this further.

6

Computing Resource Management Analyses

6.1 Introduction

Chapter 5 has demonstrated the framework's general suitability for solving different SDR computing resource management problems. The objective of this Chapter is further analyzing the capabilities of our proposal for a flexible computing resource management. We discuss the framework's diversity and examine its behavior while solving different reconfiguration problems. We, particularly, consider additional simulations and discuss the significance of the mapping order (Section 6.2), the implications of the hardware architecture (Section 6.3), and the relevance of the window size w and the cost function parameter q (Section 6.4).

6.2 Mapping Order

6.2.1 Motivation and Scope

In heterogeneous computing contexts, some tasks may have a higher importance or priority than others. The precedence constraints between the tasks of an application and the heterogeneous processing and data flow requirements furthermore indicate that the mapping or scheduling order may affect the result. Related work already demonstrated the relevance of the mapping and scheduling orders [79]. The *highest levels first with estimated times* (HLFET) algorithm [36], for instance, schedules tasks in the order of their previously computed, static priorities. These priorities, or levels, are defined as the largest sum of execution times along the directed path from the given node to an end node of the task graph. Reference [40], on the other hand, uses dynamic levels to determine the next process for mapping and scheduling. The levels of the remaining processes are thus dynamically recalculated during the task allocation process. DCP [42] or DPS [45], discussed in Section 2.4.1, follow a similar approach; many more exist [79].

Here we study static ordering techniques, where the premapping order is specified before executing the t_w-mapping algorithm. We consider the reordering of SDR functions by decreasing processing or bandwidth requirements (c-ordering or b-ordering) and analyze the mapping success as a function of the mapping order. The SDR computing system modeling of Chapter 3 facilitates such a reordering (Section 6.2.2). Simulation results show that the premapping order is relevant and indicate that the mapping order should be specified as a function of the given problem (Sections 6.2.3 and 6.2.4). A dynamic reordering of the remaining functions and data flows to be premapped (as a function of the remaining computing resources, for instance) will be examined in future work.

6.2.2 Modeling Support

We maintain the t_w-mapping process, which premaps $(f_i)^d$ before $(f_{i+1})^d$ (Section 4.2.1), and change the premapping order through relabeling: The original SDR function labels comply with the logical numbering principle (Section 3.3.2). After relabeling, 1, 2, …, $M(d)$ index SDR functions in the desired order. Reordering through relabeling is, hence, a preprocessing mapping operation.

The c- and b-ordering algorithms relabel SDR functions as a function of the processing and data flow requirements. That is, the c-ordering leads to $(c_1)^d \geq (c_2)^d \geq \ldots \geq (c_{M(d)})^d$, whereas the b-ordering results in the pair of functions with the heaviest data flow demand becoming $(f_1)^d$ and $(f_2)^d$ and those with the lowest demand $(f_{M(d)-1})^d$ and $(f_{M(d)})^d$.

The application models of Section 3.3.2B) facilitates the reordering or relabeling of SDR functions through basic matrix operations: To exchange SDR function labels $(f_u)^d$ and $(f_v)^d$, exchange $(c_u)^d$ and $(c_v)^d$ in (3.10) and switch rows and columns u and v in (3.11) or (3.12). Switching rows u and v in (3.11) or (3.12) changes the successors of $(f_u)^d$ for those of $(f_v)^d$ and vice versa, whereas switching columns u and v changes the predecessors of $(f_u)^d$ for those of $(f_v)^d$ and vice versa. This maintains the same DAG with another labeling of functions.

Fig. 6.1 illustrates the above relabeling process for a four-node DAG example, where SDR application ii corresponds to a new presentation of SDR application i. The new application models, $\boldsymbol{c}^{ii}$, $\boldsymbol{l}^{ii}$, $\boldsymbol{i}^{ii}$, and $\boldsymbol{b}^{ii}$ (Fig. 6.1d), mathematically capture the task graph of Fig. 6.1c, whereas the original models, $\boldsymbol{c}^{i}$, $\boldsymbol{l}^{i}$, $\boldsymbol{i}^{i}$, and $\boldsymbol{b}^{i}$ (Fig. 6.1b), describe the processing chain of Fig. 6.1a. Ignoring the labels, Fig. 6.1a and Fig. 6.1c show identical graphs. Hence, merely the computing resource management labels have changed: If $(f_2)^i$ represents a filter, for example, $(f_4)^{ii}$ symbolizes the same filter.

SDR platforms can be equivalent relabeled. The processor labels $(P_1)^D$, $(P_2)^D$, …, $(P_{N(D)})^D$ and the SDR function labels $(f_1)^d$, $(f_2)^d$, …, $(f_{M(d)})^d$ are thus the basis for our SDR computing system models. Any labeling and the corresponding modeling correctly characterize a given SDR platform or application. The labeling rules of Section 3.3 are appropriate for the general modeling, whereas other rules may be more appropriate for specific applications of the modeling. We analyze this in continuation.

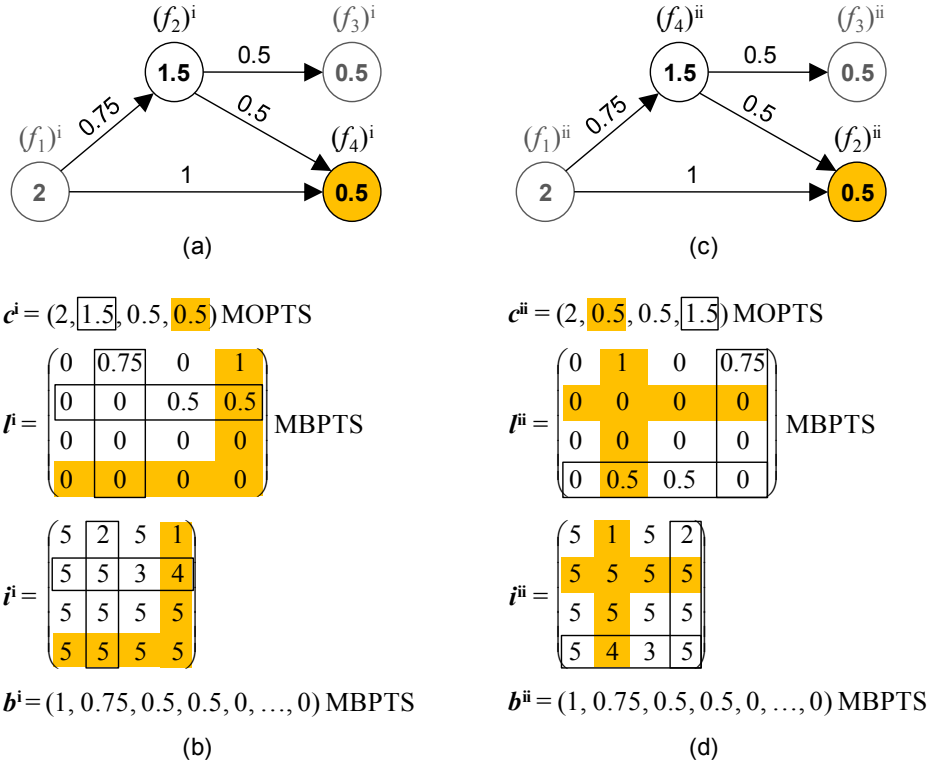


Fig. 6.1. SDR function relabeling example: example task graph (a) and its modeling (b); the resulting task graph (c) and its modeling (d) after exchanging the labels of two SDR functions.

6.2.3 Scenarios and Results, Part I

We assume the two SDR scenarios of Chapter 5. First we consider SDR scenario II (Section 5.3.1), which is based on 1 000 000 random DAGs that are individually mapped to each one of the four SDR platforms I-IV of Fig. 3.3. We compute and analyze the t_w-mapping results without reordering (no-ORD-t_w-mapping or, simply, t_w-mapping), after c-ordering (c-ORD-t_w-mapping), and after b-ordering (b-ORD-t_w-mapping). The performance metric is the percentage of infeasible mappings. Fig. 6.2a and b show the results for w = 1 and 3.

Both ordering approaches facilitate the feasible t_w-mapping of many DAGs. The c- or b-ordering, more precisely, decreases the number of infeasible mappings by a factor between two and six. The c-ordering performs better for SDR platform I, whereas the b-ordering results more suitable for platform II. For small w, platforms III and IV take advantage of the b-ORD-t_w-mapping, whereas for $w \geq 3$, the c- and b-ORD-t_w-mapping results are nearly identical (Fig. 6.2).

The bottleneck of platform II is the scarce bandwidth between $(P_1)^{II}$ and $(P_3)^{II}$ (Fig. 3.3b). The cost function's $cost_{COMP}$ term (4.9b) balances the processing load, tending to distribute the SDR functions. Since the c-ORD-t_w-mapping prioritizes the premapping of heavy processing demands, it is possible that heavy data flows may not be feasibly allocatable later on during the mapping process. The heterogeneous processing capacities of SDR platform IV (Fig. 3.3c), on the other hand, generally lead to premapping more SDR functions to $(P_1)^{IV}$, fewer to $(P_2)^{IV}$, and even fewer to $(P_3)^{IV}$; this decreases the probability of (heavy) communication demands between $(P_1)^{IV}$ to $(P_3)^{IV}$ throughout the course of the mapping and explains the results.

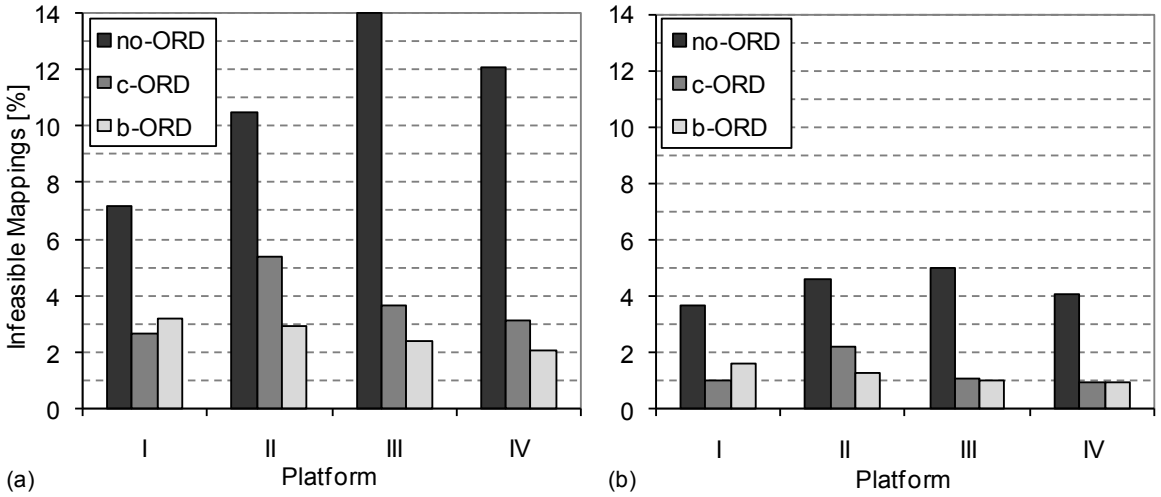


Fig. 6.2. Percentage of infeasible no-ORD-, c-ORD- and b-ORD-t_w-mappings for w = 1 (a) and w = 3 (b).

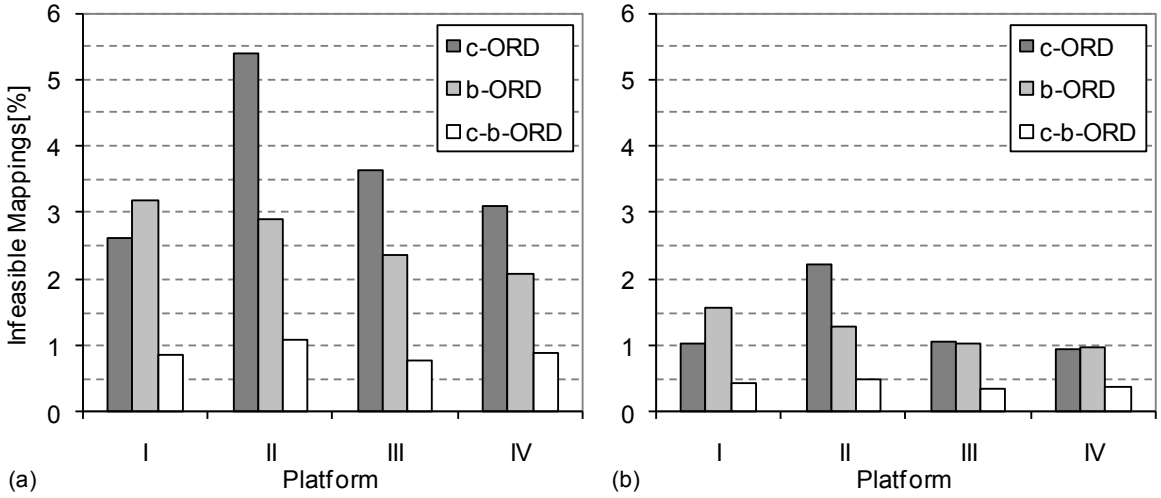


Fig. 6.3. Percentage of infeasible c-ORD-, b-ORD-, and c-b-ORD-t_w-mappings for w = 1 (a) and w = 3 (b).

We now apply the *c*-ORD-t_w-mapping first and, in case of an infeasible result, try the *b*-ORD-t_w-mapping thereafter. We call this the *c*-*b*-ORD-t_w-mapping. Fig. 6.3 shows its outcomes together with the *c*- and *b*-ORD-t_w-mapping results. We observe that the *c*-*b*-ORD-t_w-mapping can feasibly map many more DAGs than the *c*- or the *b*-ORD-t_w-mapping. Hence, the DAGs that are unfeasibly mapped with the *c*-ORD-t_w-mapping only partially overlap with those that are unfeasibly mapped with the *b*-ORD-t_w-mapping. We try to find a rule for choosing the most appropriate premapping order and, therefore, analyze some DAG statistics.

Since infeasible mappings are less frequent here than feasible mappings, the statistics of the unfeasibly mapped DAGs are probably more informative than the statistics of the feasibly mapped graphs. These statistics may reveal certain features that explain the infeasible mappings. First we compute and analyze the application's communication-to-computation ratio, defined in Table 3.10, averaged over the unfeasibly mapped DAGs. We call this parameter ccr_{inf}.

Table 6.1 and Table 6.2 provide ccr_{inf} of the *c*- and *b*-ORD-t_1-mapping results; the corresponding values for other window sizes are slightly lower. Notice that, since more than 94.5 % of the considered DAGs are feasibly mapped with the *c*- or *b*-ORD-t_1-mapping to any of the four platforms (Fig. 6.3), the CCR averaged over the feasibly mapped DAGs would approximate 0.27, the mean CCR of the random DAGs.

We observe that ccr_{inf} of the *b*-ORD-t_1-mappings is, for any platform, lower than that of the *c*-ORD-t_1-mappings. This can be explained as follows: The *c*-ORD-t_w-mapping premaps SDR functions in the order of decreasing processing requirements. It thus better handles high processing loads than heavy data flow requirements. Accordingly, the *b*-ORD-t_w-mapping, which gives priority to SDR functions with high data flow demands, can easier solve high bandwidth requirements than elevated processing loads. The processing and bandwidth loads averaged over of the unfeasibly mapped DAGs—$c\text{-}load_{inf}$ and $b\text{-}load_{inf}$, which are computed according to (5.1) with $sf_C = sf_B = 1$—confirm this (Table 6.1 and Table 6.2).

We applied different CCR-thresholds, derived from Table 6.1 and Table 6.2, for choosing one or the other ordering. None of these thresholds, however, led to results close to those of the *c*-*b*-ORD-t_w-mapping and not even improved the numbers of the *c*- or *b*-ORD-t_w-mapping. The reason for this could be that the application's communication-to-computation ratio masks the significant computing constraint: A high $(ccr)^d = (b_T)^d / (c_T)^d$ (Table 3.10) indicates a high bandwidth demand $(b_T)^d$ of SDR application or DAG *d*, although the processing resources could be more critical when the application's processing requirement $(c_T)^d$ approaches the platform's processing power $(C_T)^D$. We, therefore, consider the one-dimensional resource parameter *c*-*load*.

Since the total processing capacity is fixed—$(C_T)^D$ = 30 000 MOPS for any $D \in$ I, II, III, IV (Fig. 3.3)—a DAG's total processing requirement specifies the processing load of the corresponding mapping problem as $c\text{-}load^d = (c_T)^d / (C_T)^D$. The *b*-ordering is applied if $c\text{-}load^d$ falls below the *c*-*load*-threshold and the *c*-ordering, otherwise. We call this approach the *c*|*b*-ORD-t_w-mapping. It takes advantage of the mapping flexibility of SDR functions with low processing requirements, considering them last when processing loads are elevated. An empirical analysis led to the following *c*-*load*-thresholds: 90 % for platform I, 91.7 % for platforms II and III, and 95 % for platform IV. Fig. 6.4 shows the results.

We compare the results of Fig. 6.4 with those of Fig. 6.3 and observe that the *c*|*b*-ORD-t_w-mapping leads to fewer infeasible mappings than the *c*- or the *b*-ORD-t_w-mapping, although not reaching the numbers of the *c*-*b*-ORD-t_w-mapping. The *c*-*b*-ORD-t_w-mapping is, however, impractical as it runs twice as

Table 6.1 DAG statistics of the infeasible *c*-ORD-t_1-mappings.

Platform	ccr_{inf}	$c\text{-}load_{inf}$ [%]	$b\text{-}load_{inf}$ [%]
I	0.33	82.7	129.8
II	0.32	80.9	125.4
III	0.32	83.1	128.2
IV	0.29	88.3	124.0

Table 6.2 DAG statistics of the infeasible *b*-ORD-t_1-mappings.

Platform	ccr_{inf}	$c\text{-}load_{inf}$ [%]	$b\text{-}load_{inf}$ [%]
I	0.27	89.1	115.2
II	0.25	91.7	113.5
III	0.24	92.8	110.7
IV	0.23	94.0	108.0

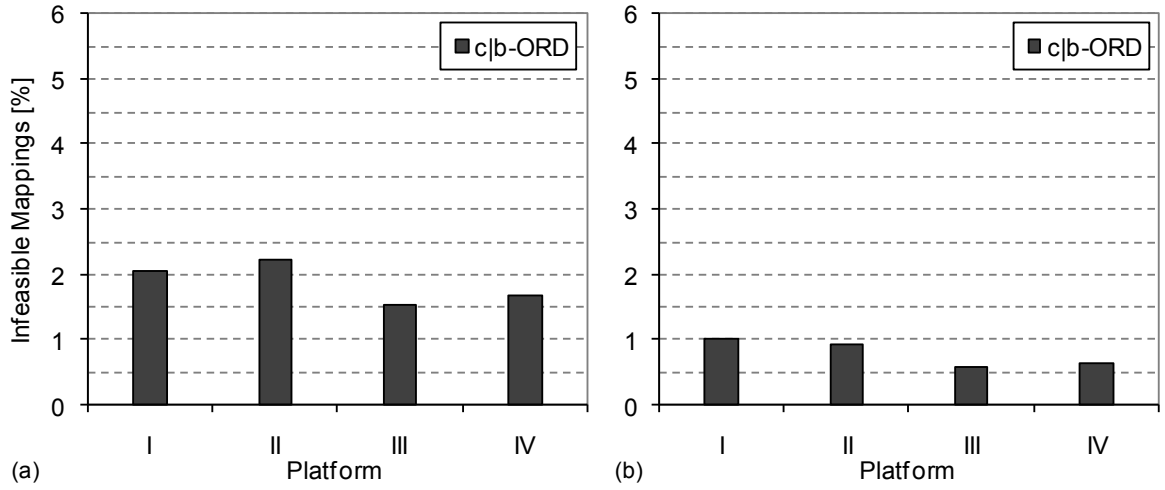


Fig. 6.4. Percentage of infeasible *c*|*b*-ORD-t_w-mappings for $w = 1$ (a) and $w = 3$ (b).

Table 6.3 ORD3- versus *c*-*b*-ORD-t_w-mapping: infeasible percentages.

	$w = 1$		$w = 3$	
Platform	*c*-*b*-ORD	ORD3	*c*-*b*-ORD	ORD3
I	0.84	0.67	0.42	0.34
II	1.09	0.88	0.50	0.42
III	0.77	0.68	0.34	0.30
IV	0.87	0.76	0.37	0.33

long for certain DAGs. The *c*|*b*-ORD-t_w-mapping is, thus, an interesting solution here and motivates for further studies (Section 6.2.4).

We finally apply the ORD3-t_w-mapping, which tries up to three mapping intents per DAG: If the t_w-mapping applied on the initial DAG model does not find a feasible solution, the *c*-ORD-t_w-mapping is executed and, if its outcome is also infeasible, the *b*-ORD-t_w-mapping is finally applied. These results, when compared with those of the *c*-*b*-ORD-t_w-mapping (Table 6.3), again confirm that the premapping order matters and that, opposed to previous conclusions, no reordering can be the better option than the *c*- or *b*-ordering.

SDR scenario I, characterized by the UMTS task graph and the four SDR platforms of varying computing resources (Section 5.2.1), leads to similar t_w-mapping results with or without a prior *b*-ordering, but worse results after the *c*-ordering. More precisely, the *b*-ORD-t_w-mapping results are very similar to those of Fig. 5.2, whereas many more mappings are infeasible when applying the *c*-ORD-t_w-mapping.

The sampling rate and, thus, the bandwidth requirements of the UMTS task graph by and large decrease with the data flow through the processing chain (Fig. 3.4 and Fig. 3.7a). Hence, the original labeling of SDR functions (Fig. 3.7a) only slightly differs from that after the *b*-ordering. This explains the similar results with or without *b*-ordering. The worse performance of the *c*-ORD-t_w-mapping can be explained as follows: The heavy data flows between some SDR functions in comparison with the interprocessor bandwidth capacities of any SDR platform make it important to execute these functions on a single processor. Considering them at a later stage of the *c*-ORD-t_w-mapping process, may result in the remaining processing resources per processor being insufficient for joining heavily communicating functions.

We summarize that both ordering technique considerably improve the mapping results of SDR scenario II, whereas the *c*-ordering is not suitable for scenario I. Hence, the selection of the appropriate ordering is case dependent. Motivated by the above results, the second part of this study statistically examines the relations between the processing and bandwidth loads and the ordering selection.

6.2.4 Scenario and Results, Part II

Here we reconsider the scenario of Section 5.3.1 but vary the DAG generation parameters. We generate four times one million DAGs of $M(d) = 18$ and $con^d = 0.15$ (Section 5.3.1), while considering the following variations of the processing and bandwidth requirements:

(A) $(c_u)^d \in \{1, 2, ..., 2100\}$ MOPS and $(l_{uv})^d \in \{1, 2, ..., 400\}$ Mbps,

(B) $(c_u)^d \in \{1, 2, ..., 2100\}$ MOPS and $(l_{uv})^d \in \{1, 2, ..., 600\}$ Mbps,

(C) $(c_u)^d \in \{1, 2, ..., 2700\}$ MOPS and $(l_{uv})^d \in \{1, 2, ..., 400\}$ Mbps, and

(D) $(c_u)^d \in \{1, 2, ..., 2700\}$ MOPS and $(l_{uv})^d \in \{1, 2, ..., 600\}$ Mbps.

These four setups can be characterized by the mean processing and bandwidth loads $c\text{-}load_M$ and $b\text{-}load_M$, which correspond to *c-load* and *b-load* of (5.1) with $sf_C = sf_B = 1$ averaged over all considered DAGs. (We discard those DAGs that require more processing resources than available.) We empirically obtain $c\text{-}load_M$ and $b\text{-}load_M$ and qualify the four scenarios as

(A) ($c\text{-}load_M$, $b\text{-}load_M$) = (0.63, 0.80) – low processing and bandwidth loads,

(B) ($c\text{-}load_M$, $b\text{-}load_M$) = (0.63, 1.20) – low processing and high bandwidth loads,

(C) ($c\text{-}load_M$, $b\text{-}load_M$) = (0.80, 0.80) – high processing and low bandwidth loads, and

(D) ($c\text{-}load_M$, $b\text{-}load_M$) = (0.80, 1.20) – high processing and bandwidth loads.

Fig. 6.5 to Fig. 6.8 show the infeasible t_w-mapping, c-ORD-t_w-mapping, and b-ORD-t_w-mapping percentages for the four scenarios (A) to (D). The first conclusion is that the mapping challenge is a function of ($c\text{-}load_M$, $b\text{-}load_M$): Relatively low mean processing and bandwidth loads (A) are easily solvable. Hence, the *c*-ordering and the *b*-ordering are both appropriate, although directly applying the t_1-mapping already leads to acceptable results with more than 98.3 % feasibly mapped DAGs (Fig. 6.5). A high $b\text{-}load_M$ and a low $c\text{-}load_M$ (B) suggest the *b*-ordering (Fig. 6.6), whereas the *c*-ordering should be considered for the opposite case (C) due to Fig. 6.7. When the $c\text{-}load_M$ and the $b\text{-}load_M$ are both elevated (D),

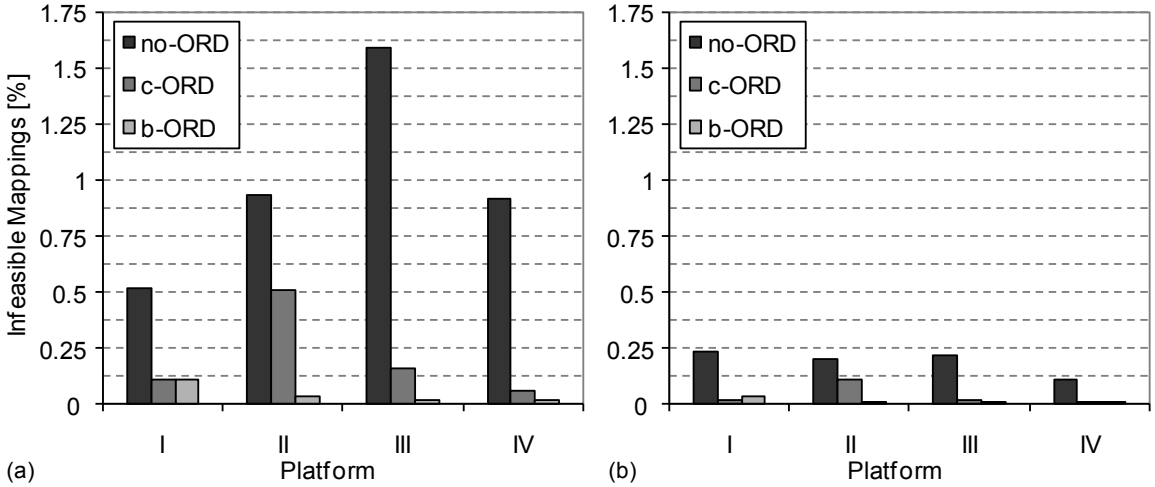


Fig. 6.5. no-ORD-, *c*-ORD-, and *b*-ORD-t_w-mapping results for $w = 1$ (a) and $w = 3$ (b) for scenario (A).

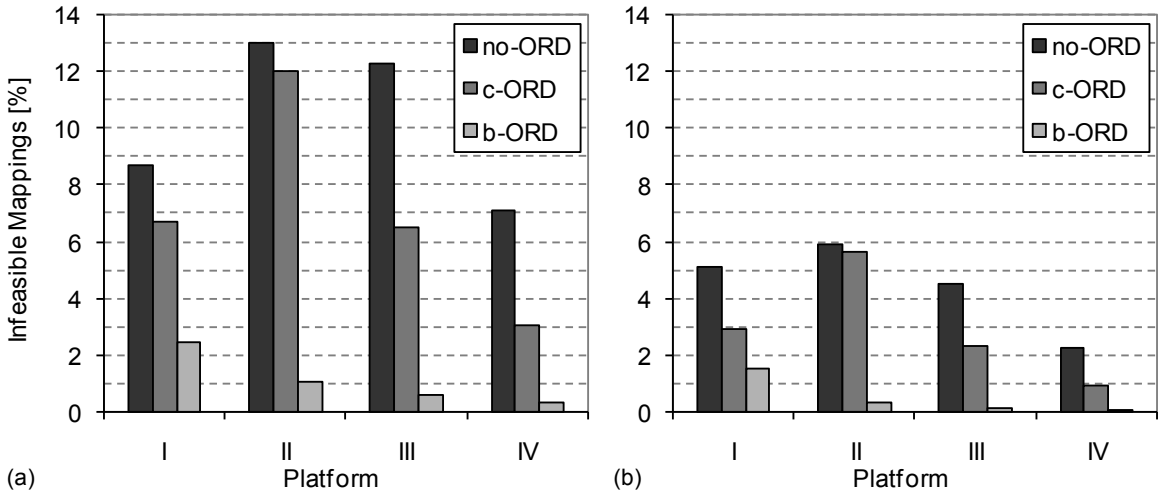


Fig. 6.6. no-ORD-, *c*-ORD-, and *b*-ORD-t_w-mapping results for $w = 1$ (a) and $w = 3$ (b) for scenario (B).

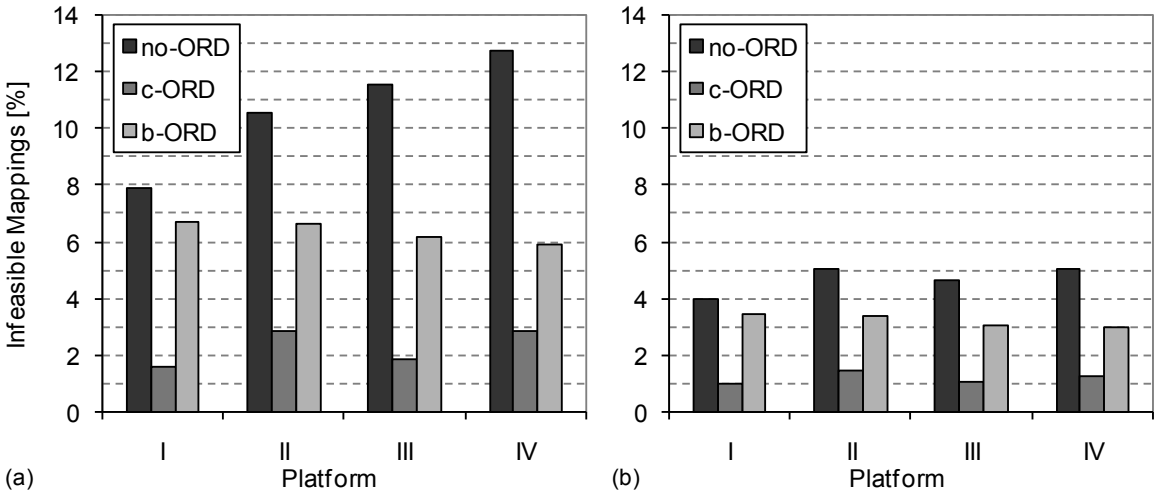


Fig. 6.7. no-ORD-, *c*-ORD-, and *b*-ORD-t_w-mapping results for $w = 1$ (a) and $w = 3$ (b) for scenario (C).

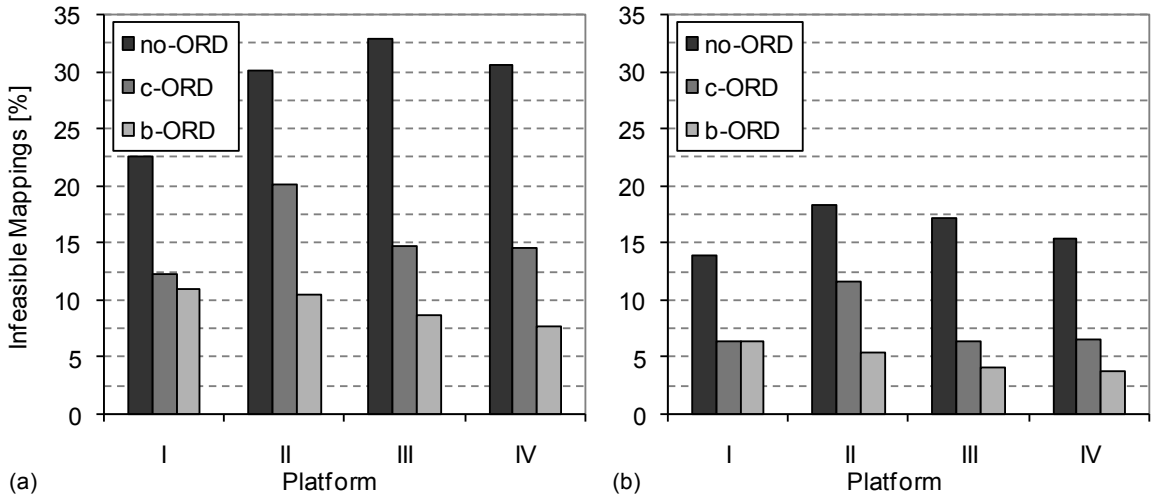


Fig. 6.8. no-ORD-, *c*-ORD-, and *b*-ORD-t_w-mapping results for $w = 1$ (a) and $w = 3$ (b) for scenario (D).

the ordering decision depends on the particular problem. Here, the *c*-ORD- and the *b*-ORD-t_w-mapping perform well for SDR platform I, whereas the *b*-ORD-t_w-mapping is more suitable for platforms II, III, and IV (Fig. 6.8). Both ordering approaches improve the no-ORD-t_w-mapping results for any scenario (Fig. 6.5-Fig. 6.8).

These results indicate the suitability of the four simple parameters $(c_T)^d$, $(b_T)^d$, $(C_T)^D$, and $(B_T)^D$. Despite the specific simulation setup, they provide some general tendencies or indications, which we summarize as

- low *c-load* and low *b-load*: apply the *c*-ordering or the *b*-ordering,
- low *c-load* and high *b-load*: apply the *b*-ordering,
- high *c-load* and low *b-load*: apply the *c*-ordering, and
- high *c-load* and high *b-load*: apply the *c*-ordering or the *b*-ordering (case dependent).

6.3 Hardware Architecture

6.3.1 Motivation and Scope

Reconfigurability is the key concept of SDR. It indicates that one and the same computing platform can, in different configurations, be used for transmitting and receiving data over different air interfaces (frequen-

cy bands and modulation schemes). The radio transceiver software (waveform or SDR application) that runs on an SDR platform thus specifies the momentarily adopted radio standard.

The reconfiguration of SDR platforms may be partial or total: A total reconfiguration deinstalls the entire software, freeing all computing resource, before installing the new one. A partial reconfiguration, on the other hand, may deinstall part of the software while installing new software pieces. Hence, the available computing resources of SDR platforms may vary from configuration to configuration. Furthermore, some platforms may facilitate the integration of plug-and-play hardware. Most SDR-BS will be designed with such an upgradability feature; SDR-MT may eventually follow.

SDR platforms will differ in computing architectures and capabilities. Platforms may be distinguishable by the number of processors, the processing capacities, the interprocessor communication network, and the communication bandwidths, among others. A platform's initial computing capabilities may change over time so that, before initiating a partial reconfiguration, the available computing architecture and resources (*platform state*) may not match the architecture and resources of the platform in its initial state (*platform architecture*). Regarding the computing resource management, there is no difference between platform architecture (initial computing architecture and resources) and platform state (momentary available computing architecture and resources).

The vast amount of possible platform architectures and states motivates analyzing the interrelation between platform architecture and application mapping. Here we consider a very limited set of SDR platform architectures or states while examining the implications of the interprocessor communication network (Section 6.3.2) and the distribution of computing resources (Section 6.3.3).

6.3.2 Connectivity and Communication Flexibility

A) Scenario – Simulation Setup

This scenario considers five SDR platforms. Fig. 6.9 illustrates their architectures and computing resources. Dedicated FD links characterize SDR platforms IX (Fig. 6.9a) and XII (Fig. 6.9d), dedicated HD buses describe the interprocessor communication networks of platforms X (Fig. 6.9b) and XIII (Fig. 6.9e), whereas a shared HD bus characterizes platform XI (Fig. 6.9c). The total processing and bandwidth resources per platform are 9000 MOPS and 9000 Mbps. Fig. 6.10 contains the relevant parameters for distinguishing these platforms from one another.

For avoiding a particular implementation and providing statistically representative results, we generate 1 000 000 random DAGs (Section 5.3.1) based on the following parameters:

- $M(d) = 25$,
- $con^d = 0.2$,
- $(c_u)^d$ ($u \in 1, 2, \ldots, M(d)$) uniformly distributed in {1, 2, ..., 500} MOPS, and
- $(l_{uv})^d$ ($u, v \in 1, 2, \ldots, M(d)$) of existing data flow connections uniformly distributed in {1, 2, ..., 500} Mbps.

These DAGs represent different SDR application models that require

$$(c_T)^d = 25 \cdot (1 \text{ MOPS} + 500 \text{ MOPS}) / 2 = 6262.5 \text{ MOPS} \tag{6.1}$$

in the mean, which is about 70 % of the available processing capacity. ($(c_T)^d$, the sum of 25 uniformly distributed independent random variables, approximately follows the Gaussian probability density function of 6262.5 mean and 720 standard deviation.) The total processing requirement is higher than 4500 and 9000 MOPS with a probability of 0.99 and $5 \cdot 10^{-5}$, respectively. Those 51 DAGs that require more processing resource than available are discarded.

Almost all DAGs require more bandwidth than the 9000 Mbps that are available for interprocessor data flows. (This requires solving some data flows processor-internally by allocating the same processor to the corresponding SDR functions.) Moreover, the statistically expected value of the total bandwidth demand is approximately

$$E[(b_T)^d] \approx (con^d \cdot (M(d)^2 - M(d)) / 2) \cdot (500 \text{ Mbps} + 1 \text{ Mbps}) / 2 = 15\,030 \text{ Mbps}, \tag{6.2}$$

which is 167 % of the available 9000 Mbps, indicating a significant bandwidth resource constraint.

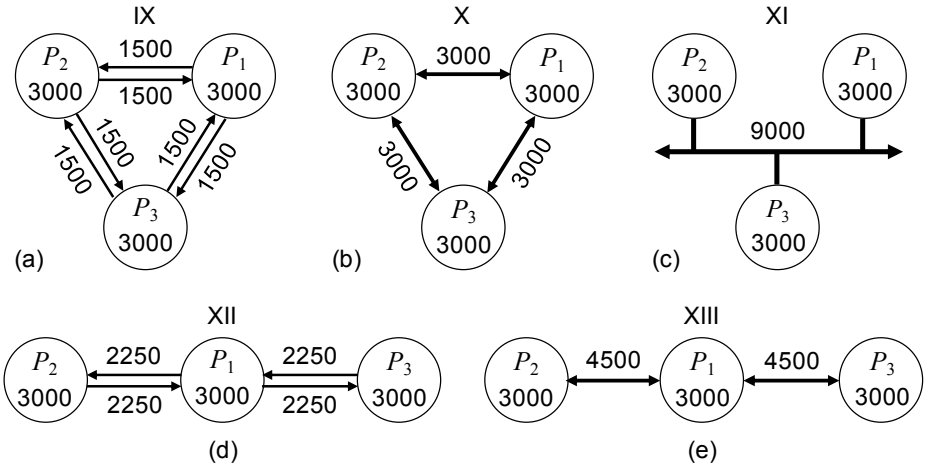


Fig. 6.9. SDR platforms IX-XIII (a)-(e) and their computing resources in MOPS and Mbps.

Again we directly specify the time slot duration t_{TS} as $0.5 \cdot 10^{-3}$ SPTS. $M(d) \cdot t_{TS} = 12.5$ ms will then be the maximum latency of any of these SDR applications due to (3.15) and Table 3.9. The mapping order is not significant for this and the following analyses. We apply the *c*-ordering in the rest of this chapter because of its simple implementation and good performance (Section 6.2).

B) Simulation Results

The performance metric is the percentage of infeasible mappings. Fig. 6.11a shows the results for SDR platforms IX, X, and XI. We observe that SDR platform XI leads to considerably fewer infeasible mappings than platform X, which behaves better than platform IX. Fig. 6.10 indicates that the number of infeasible mappings decreases as the communication flexibility, represented through the mean link flexibility LF_M, increases. Fig. 6.11b illustrates this for the *c*-ORD-t_1-mapping.

SDR platforms XII and XIII, which have a lower connectivity than platforms IX and X (Fig. 6.10), have a low *c*-ORD-t_w-mapping success (Fig. 6.12a and b). Fig. 6.12c presents the *c*-ORD-t_1-mapping results as a function of the connectivity *CON* and the mean link flexibility LF_M. We observe that the higher *CON* or LF_M the better the mapping performance. We conclude that the mapping success is a function of *CON* and LF_M and that highly and flexibly interconnected SDR platforms, such as XI, ease the allocation of distributed computing resources.

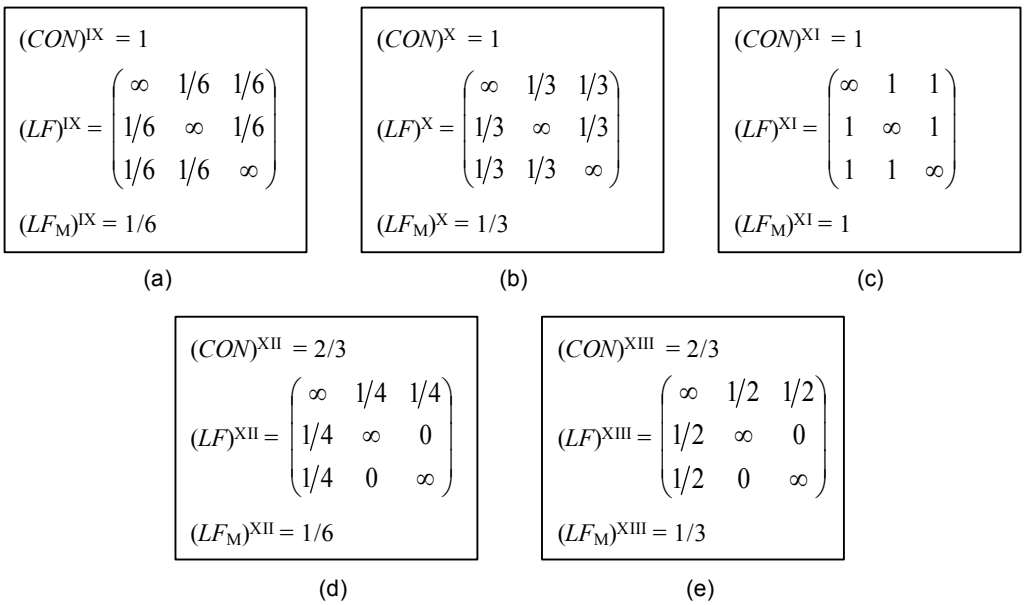


Fig. 6.10. Selected communication parameters of SDR platforms IX-XIII (a)-(e).

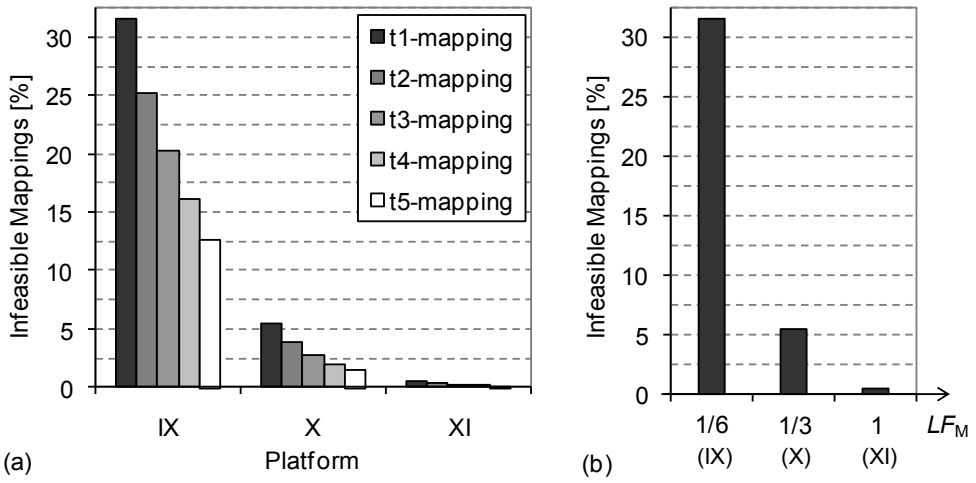


Fig. 6.11. Percentage of infeasible *c*-ORD-t_w-mappings for SDR platforms IX, X, and XI as a function of w (a) and as a function of LF_M for $w = 1$ (b).

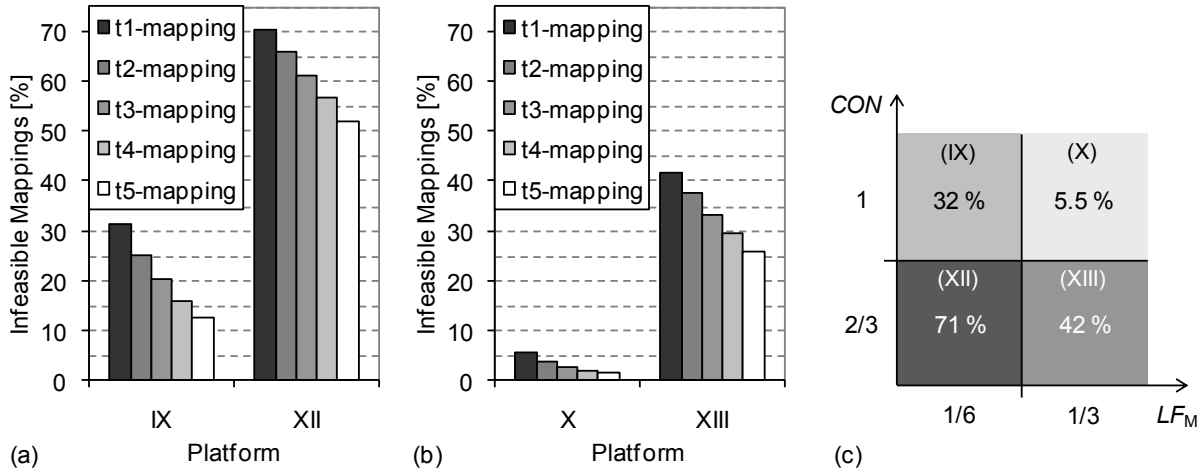


Fig. 6.12. Percentage of infeasible *c*-ORD-t_w-mappings for SDR platforms IX, XII, X, and XIII as a function of w (a), (b) and as a function of LF_M and CON for $w = 1$ (c).

6.3.3 Distribution of Computing Resources

A) Scenario – Simulation Setup

This scenario assumes that the partial deallocation of resources immediately before their reallocation leaves the SDR platform in one of the nine states of Fig. 6.13. A platform state, or architecture, here represents the momentarily available computing resources of a given SDR platform.

Homogeneous processing and bandwidth capacities characterize platform state XIV, heterogeneous processing capacities state XV, heterogeneous bandwidth capacities state XVI, and heterogeneous processing and bandwidth capacities states XVII-XXII (Fig. 6.14). Any platform state D ($D \in$ XIV, XV, …, XXII) provides a total processing capacity of 9000 MOPS and a total interprocessor bandwidth of 12 000 Mbps. A platform's communication-to-computation ratio CCR^D is 1.333. Fig. 6.14 resumes the characteristic platform parameters.

Since analyzing the implications of the hardware architecture, we cannot assume a specific SDR application. We, therefore, reconsider the 1 000 000 random DAGs of Section 6.3.2A). The SDR scenario is the following: The nine platform states of Fig. 6.13 represent the available computing resources at some instant, be it the initial states of nine SDR-MTs, after several partial reconfigurations, or after a recent hardware upgrade. Each one of these states is reconfigured 1 000 000 times, mapping and completely demapping a random DAG. These reconfigurations could correspond to dynamic mode switches of different SDR-MTs.

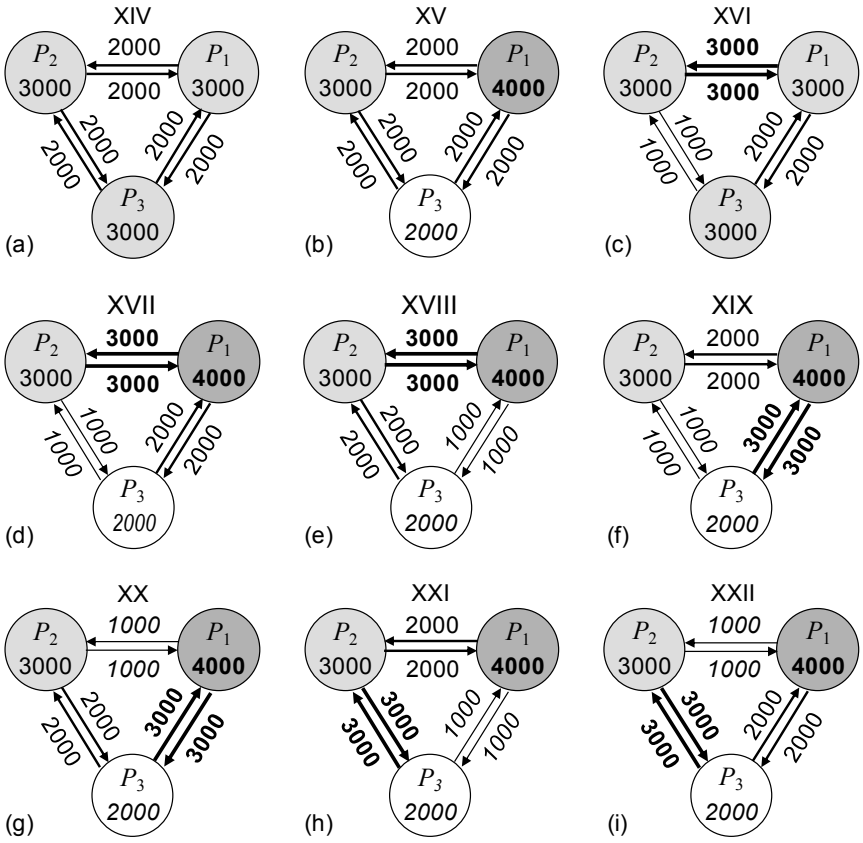


Fig. 6.13. SDR platform (states) XIV-XXII (a)-(i).

B) Simulation Results

We apply the *c*-ORD-t_w-mapping with cost function (4.9) and, again, directly specify the time slot duration t_{TS} as $0.5 \cdot 10^{-3}$ SPTS. The cost function is a superposition of the computation cost $cost_{COMP}$ and the communication cost $cost_{COMM}$, where parameter $\boldsymbol{q} = (q_1, q_2) = (q, 1–q)$ weights the two cost terms (Section 4.3.2). This analysis consists of repeating the mapping procedure for each platform and a set of applications (1 000 000 random DAGs) with different q instances. Fig. 6.15 shows the percentage of unfeasibly mapped DAGs as a function of platform state D and cost function parameter q ($q \in 0, 0.05, 0.1, \ldots, 1$) for the *c*-ORD-$t_1$-mapping (Fig. 6.15a) and the *c*-ORD-t_3-mapping (Fig. 6.15b).

First of all we observe a relation between the number of infeasible allocations and the platform state: Platform state XIV performs well because the homogeneous processors and communication links facilitate the distribution of the SDR application components. State XVI lacks the homogeneous communication network, complicating such a distribution.

Most of the processing load is likely to be distributed between $(P_1)^D$ and $(P_2)^D$ ($D \in$ XV, XVII, XVIII, …, XXII). States XVII and XVIII are favorable, because $(B_{12})^{XVII} = (B_{21})^{XVII} = (B_{12})^{XVIII} = (B_{21})^{XVIII} = 3000$ Mbps, whereas the corresponding bandwidths of XV, XIX, and XXI (XXII and XX) are merely 2000 (1000) Mbps. $(P_1)^{XXI}$ and $(P_1)^{XXII}$ have inferior communication capabilities (quantified as the sum of the processor's input and output resources) than $(P_1)^D$ of any other platform state. The fact that $(P_1)^D$ generally executes more SDR functions than $(P_2)^D$ or $(P_3)^D$ explains the poor mapping performance of platform states XXI and XXII. We should therefore try to avoid these two platform states in practice.

Parameter $(X_{STD})^D$, the platform's communication-to-computation incoherence, formally explains the above observations. It represents the standard deviation of the upper-diagonal elements of the communication-to-computation correlation matrix $\boldsymbol{X^D}$ (Table 3.5). Fig. 6.14 provides $\boldsymbol{X^D}$ and $(X_{STD})^D$ for the nine platform states.

The six platform architectures XVII-XXII (Fig. 6.13d-i) are equivalent except for the distribution of the bandwidth capacities (Fig. 6.14d-i). The corresponding mapping results indicate that the mapping success is a function of X_{STD}: the lower X_{STD} the fewer the number of infeasible mappings. $(X_{STD})^D$ then qualifies the distribution of the platform's available processing and bandwidth resources.

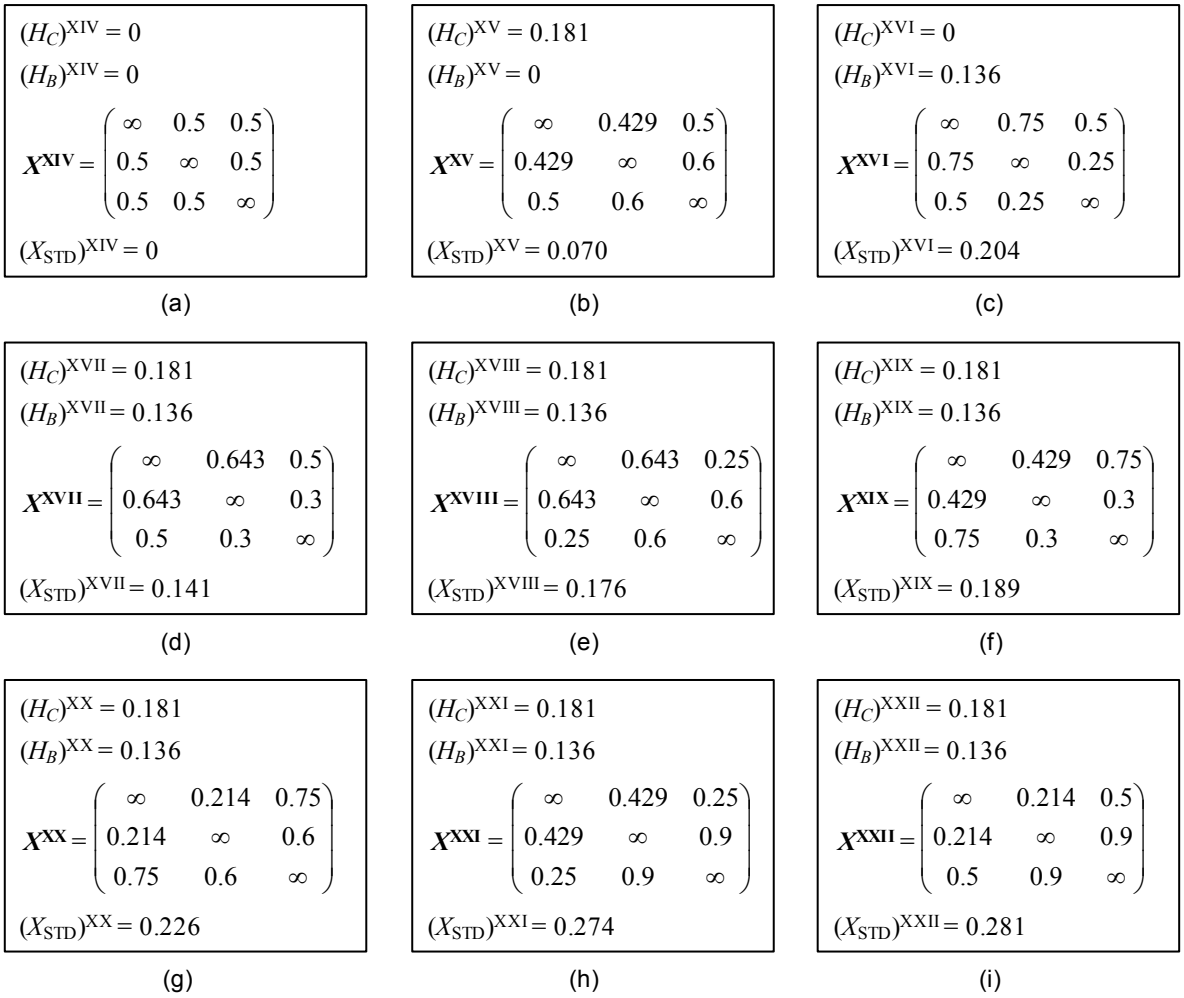


Fig. 6.14. Characteristic platform parameters of SDR platforms XIV-XXII (a)-(i).

Fig. 6.15 shows that the curves that correspond to platform states XXI and XXII are close to one another and notably separated from the other curves. Correspondingly, $(X_{STD})^{XXI}$ and $(X_{STD})^{XXII}$ are similar and higher than the communication-to-computation incoherence of any other platform state (Fig. 6.14). Architectures XVI, XIX, and XX form another group, and XIV, XV, XVII, and XVIII a third (Fig. 6.14 and Fig. 6.15).

Fig. 6.15 also indicates that the optimal q instance is a function of the SDR platform but practically independent of w (Table 6.4). Taking into account that the higher q the more decisive $cost_{COMP}$ and the lower q the more decisive $cost_{COMM}$, we can explain Table 6.4 as follows.

The processing power distribution distinguishes platform state XV from XIV (Fig. 6.13a and b). Comparing the corresponding mapping results (Fig. 6.15), we conclude that the distribution of MOPS of platform state XV makes it harder to find a feasible mapping. This explains $(q_{OPT})^{XV} > (q_{OPT})^{XIV}$. Similarly, platform state XVI, which differs from XIV in the distribution of bandwidths (Fig. 6.13a and c), shows worse mapping behavior than platform state XIV; therefore, $(q_{OPT})^{XVI} < (q_{OPT})^{XIV}$. Platforms states XVII to XXII are a combination of XV and XVI (Fig. 6.13). Hence, their optimal q instances are approximately between $(q_{OPT})^{XVI} \approx 0.45$ and $(q_{OPT})^{XV} \approx 0.65$ (Table 6.4).

Table 6.4 Optimal q instances $(q_{OPT})^D$.

w	XIV	XV	XVI	XVII	XVIII	XIX	XX	XXI	XXII
1	0.60	0.65	0.45	0.60	0.65	0.45	0.55	0.65	0.60
2	0.60	0.65	0.45	0.60	0.65	0.45	0.55	0.65	0.60
3	0.60	0.65	0.45	0.55	0.60	0.40	0.50	0.60	0.60
4	0.55	0.60	0.40	0.60	0.60	0.40	0.50	0.60	0.55
5	0.55	0.65	0.40	0.55	0.60	0.40	0.50	0.60	0.55

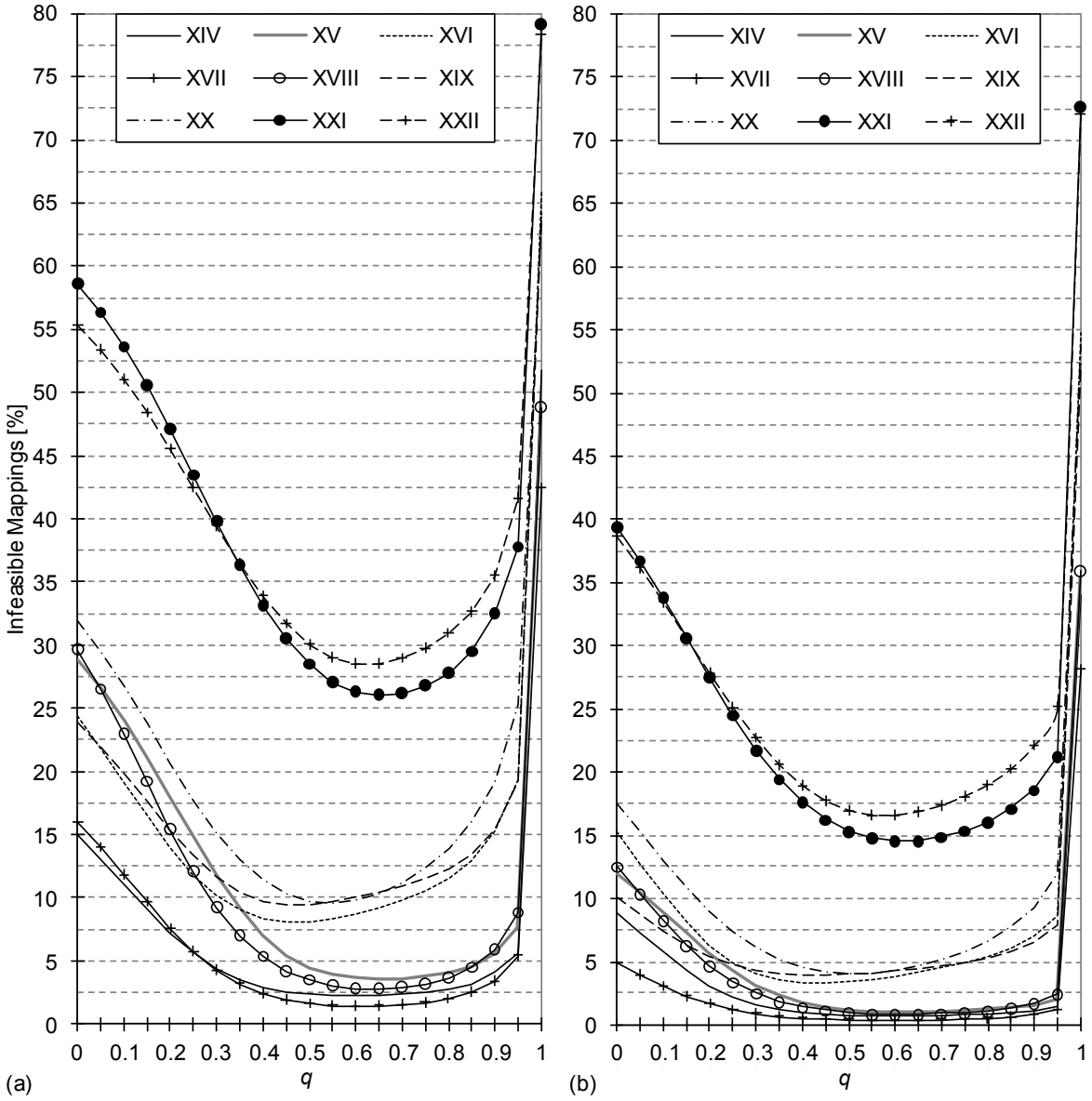


Fig. 6.15. *c*-ORD-t_1-mapping (a) and *c*-ORD-t_3-mapping results (b) for SDR platforms XIV-XXII.

The two local maxima of the 18 curves in Fig. 6.15a and b confirm that the limited processing and bandwidth capacities require a composite load balancing; that is, $0 < q < 1$. The global maximum is at $q = 1$. For $q = 1$, the cost function balances the processing load without trying to minimize (excessive) interprocessor data flows. Hence, interprocessor bandwidths are the major bottleneck of this study. Additional measurement points between $q = 0.95$ and $q = 1$ (Fig. 6.16) reveal smooth curve progressions throughout the entire q range.

We finally study the robustness of the t_w-mapping against variations of q and, therefore, compute the range of q instances (*q-range*) with no more than 10 000 additional infeasible mappings with respect to the optimal result. That is, if the optimal result for platform state D is x %, then all instances of q with less than $(x + 1)$ % infeasible allocations specify the platform's *q-range*.

Fig. 6.17 illustrates the different *q-ranges*. It indicates that that *q-range* is a function of platform state D and window size w and that all *q-ranges* are contiguous. We observe that the higher w the more robust the mapping algorithm against variations of q. In other words, the importance of q decreases with increasing w.

Since the results for $q = 0.6$ are within 1 % of the minimum number of infeasible mappings for any platform state (Fig. 6.17), the information about the optimal q instances of Table 6.4 is not very relevant here. Finding the optimal q instance could, though, be very relevant in computing resource management

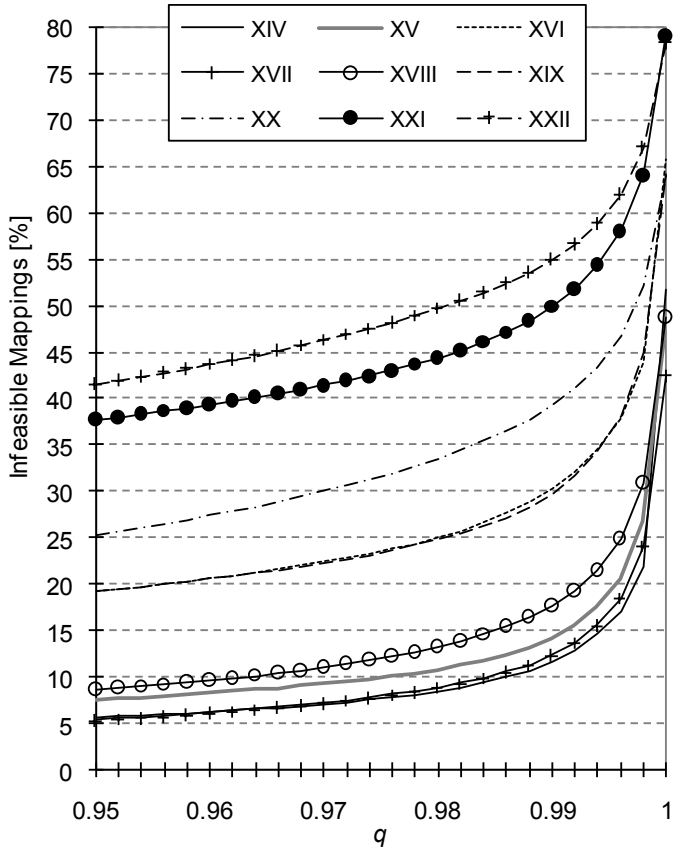


Fig. 6.16. *c*-ORD-t_1-mapping results for q = 0.95, 0.952, ..., 1.

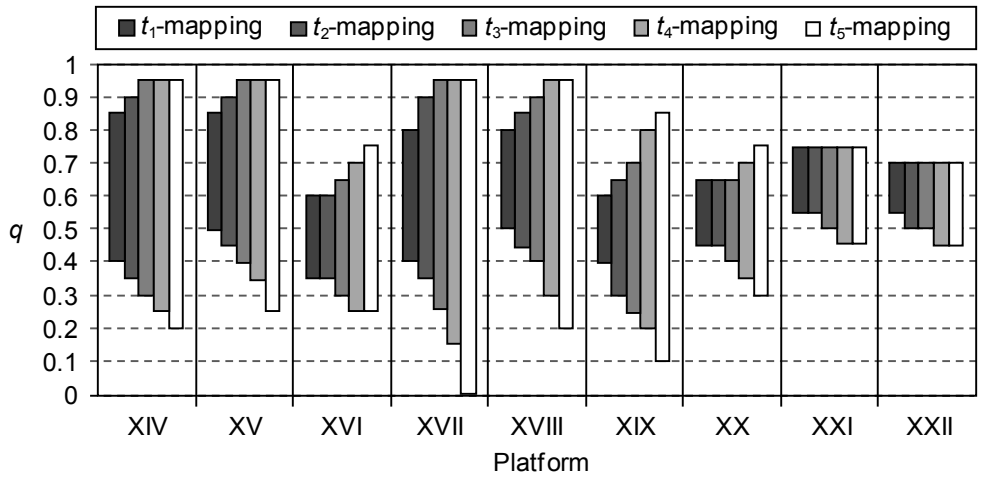


Fig. 6.17. The *q-ranges* of the *c*-ORD-t_w-mapping results.

scenarios that present a higher dependency on q. We do not study this further, but, rather, analyze the significance of parameters w and q on application basis.

6.4 Window Size versus Cost Function Parameter

6.4.1 Motivation and Scope

So far we have analyzed the importance of the mapping order, the significance of the window size w, and the relationship between the cost function parameter q and the platform architecture or state D. We have observed that the t_w-mapping performance is a function of the mapping order. All simulations have shown

that the higher w the fewer the number of infeasible mappings. We have also demonstrated the suitability of the two-term cost function (4.9) and concluded that an optimal q instance exists for any SDR platform (state).

It remains to study the significance of q on SDR application basis as well as its importance relative to w. In order to find these implications, this section applies two computing resource management methods, one considering a single and the other several q values per reconfiguration (Section 6.4.2). We examine the performances of these methods (Section 6.4.3) and analyze their complexities (Section 6.4.4).

6.4.2 Scenario and Methods

The scenario is the same as in Section 6.3.3A): 1 000 000 random DAGs are mapped to each one of the nine platform states of Fig. 6.13. We apply the c-ORD-t_w-mapping for facilitating the comparison with previous results, but, first, introduce two mapping methods.

Method A applies the c-ORD-t_w-mapping for a fixed $\boldsymbol{q} = (q, 1–q)$. In Section 6.3.3B) we have observed that $q = 0.6$ leads to (close to) optimal results for any of the nine platform states and, therefore, consider $\boldsymbol{q} = (0.6, 0.4)$ for *method A*. This method corresponds to the mapping approach of Section 6.3.3 with $q = 0.6$.

Method B, on the other hand, dynamically selects q on application basis. It repeatedly executes the t_w-mapping with different q instances until either finding a feasible result for a given DAG or having tried the entire set of instance. A set of nine instances are considered in the following order: 0.5, 0.6, 0.4, 0.7, 0.3, 0.8, 0.2, 0.9, 0.1. The extreme values $q = 0$ and $q = 1$ are discarded because of the results of Fig. 6.15. *Method B* thus examines up to nine different q values per DAG, starting with $q = 0.5$ and finishing with $q = 0.1$, if necessary. A feasible mapping immediately stops the mapping process for the corresponding DAG, whereas an infeasible mapping repeats the t_w-mapping process with another q instance. The mean number of mapping iterations per DAG (*m-iter*) is then between 1 and 9.

6.4.3 Simulation Results

Fig. 6.18 shows the simulation results as a function of D and w for both computing resource management approaches. Each subfigure corresponds to one of the nine platform states of Fig. 6.13. Since the objective here is comparing the performances between *method A* and *method B* (for each platform state individually), each subfigure may have another scale.

We observe that *method B* is capable of considerably decreasing the number of infeasible mappings. More precisely, the numbers of infeasible mappings due to *method A*, which examines a single and fixed q for all 1 000 000 reconfiguration tasks, more than doubles the corresponding results of *method B*, which tries up to nine different q instances per reconfiguration task. The particular gain is a function of the platform state and window size. The ratio between the amount of unfeasibly mapped DAGs of *method A* and *method B* is as low as 2.3 (D = XXII, w = 1; Fig. 6.18i) and as high as 20.3 (D = XVI, w = 5; Fig. 6.18c). These results indicate that parameter q is very relevant and that it should be adjusted as a function of the SDR platform and application.

Additional simulations have revealed that there is often more than one q instance that leads to a feasible t_w-mapping solution for a given problem, ordering, and window size. The problem complexity, however, makes it hard to find general rules for selecting an appropriate q as a function of the SDR application and platform (state). We, rather, analyze the processing complexities of *method A* and *method B* and introduce a metric that relates mapping performance to algorithm complexity for evaluating the importance of q relative to w.

6.4.4 Performance versus Complexity Analysis

This section evaluates the methods in terms of performance versus complexity. We therefore relate the quality of the computing resource management approach to its processing complexity:

$$\textit{metric-I} = \textit{quality} \,/\, \textit{complexity} \,. \tag{6.3}$$

The above metric may be considered an efficiency indicator, because efficiency indicates good results with low processing efforts.

Here we define *quality* as follows: If the algorithm fails in mapping x % of the applications, its *quality* is $1/x$. We measure the complexity in two ways, theoretically and practically. In both cases we consider the number of MACs as the complexity indicator. The theoretical complexity of *method A* is given by (4.25). The theoretical complexity of *method B* is obtained as (4.25) multiplied by the mean number of mapping iterations *m-iter*.

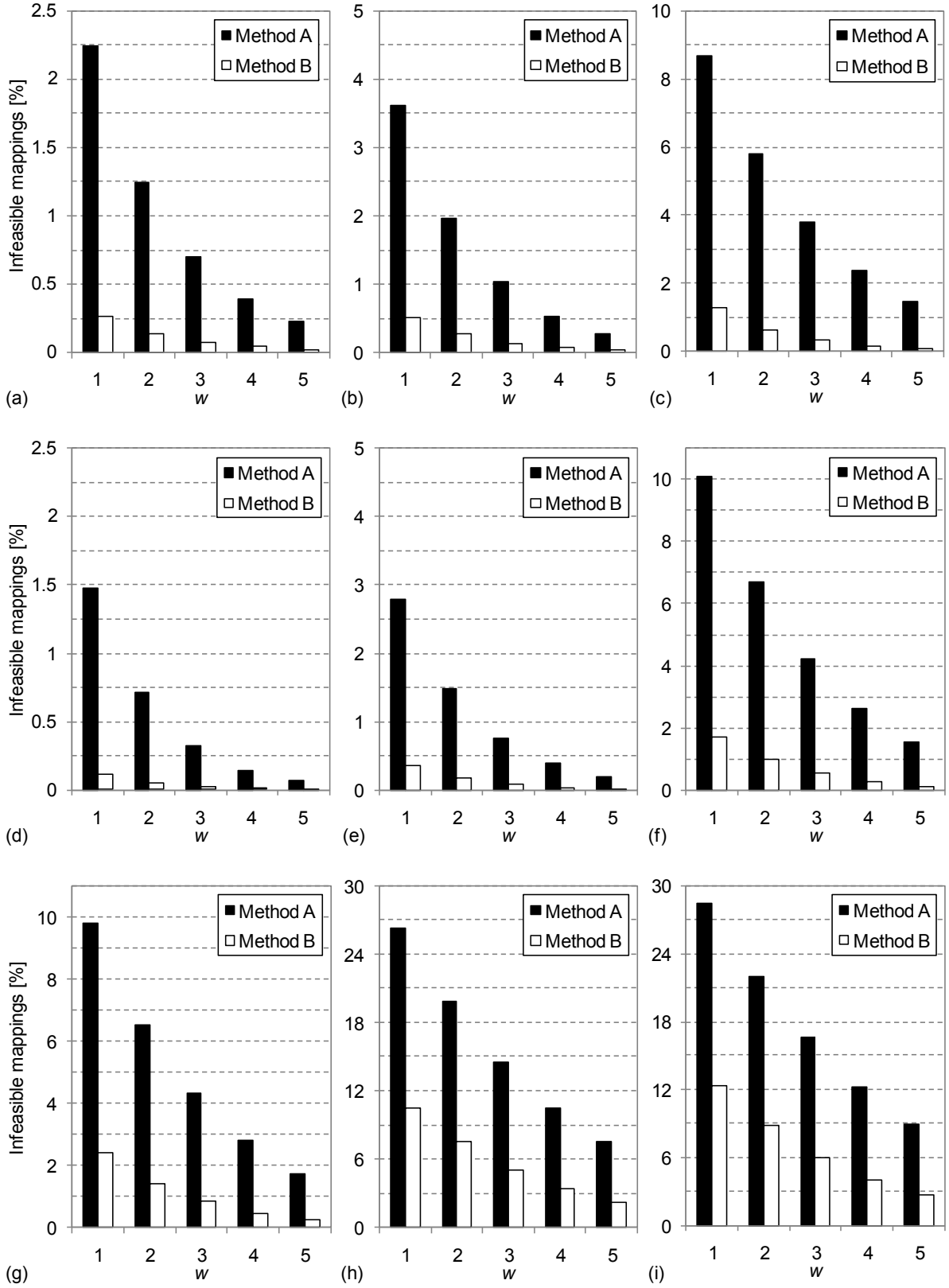


Fig. 6.18. Infeasible c-ORD-t_w-mappings due to *method A* and *method B* for platform states XIV-XXII (a)-(i).

Table 6.5 Theoretical and practical complexities of *method A* for platform state XIV.

w	Theoretical complexities (TC)	Mean practical complexities (PC)	PC / TC [%]
1	3 348	991	29.6
2	12 627	3 752	29.7
3	39 897	11 557	29.0
4	120 204	33 562	27.9
5	356 310	95 246	26.7

Table 6.6 Theoretical and practical complexities of *method B* for platform state XIV.

w	Theoretical complexities (TC)	Mean practical complexities (PC)	PC / TC [%]
1	3 581	1 056	29.5
2	13 092	3 873	29.6
3	40 706	11 739	28.8
4	121 567	33 788	27.8
5	358 591	95 399	26.6

The practical complexities are obtained from C-code implementations, counting the MACs that are actually carried out (without distinguishing between multiplication and division). Hence, the practical complexities account for code optimizations. We discard the complexity of the *c*-ordering process, executed once for each DAG.

The practical complexity numbers are between 25 and 30 % of the theoretical results, as Table 6.5 and Table 6.6 indicate for platform state XIV. The other platform states are characterized by similar numbers. The relation between the practical and theoretical complexities decreases with increasing *w*, because the code optimizations (break statements) have a higher effect for longer *w*-paths (Section 5.2.3).

Table 6.7 presents *m-iter* as a function of *w* and *D*. We observe that *m-iter* decreases with increasing *w* and that it takes the lowest values for SDR platforms XIV, XV, XVII, and XVIII. It is a function of the number of infeasible mappings (cp. Fig. 6.15 and Table 6.7).

Fig. 6.19 shows *metric-I* for platform state XIV and both methods based on the theoretical (Fig. 6.19a) and practical complexity figures (Fig. 6.19b). The different scales in the two subfigures facilitate observing that *metric-I* is qualitatively equivalent for both approaches. This fosters the theoretical complexity analysis of Section 4.4.2. We have made the same observations for the other eight platform states. Fig. 6.20 shows the results for platform states XV to XXII based on the practical complexity numbers.

Except for platform architecture XXII, *method B* has a higher *metric-I* than *method A* for any *w*; the relative difference is a function of *w* and *D*, but is, mostly, significant. This difference augments with increasing *w*, although *metric-I* decreases. This means that the complexity increase that is associated with augmenting *w* is higher than the corresponding performance gain due to *metric-I*. It is, thus, more efficient to search for a suitable *q* (*method B*) than to increase *w* (Fig. 6.19 and Fig. 6.20).

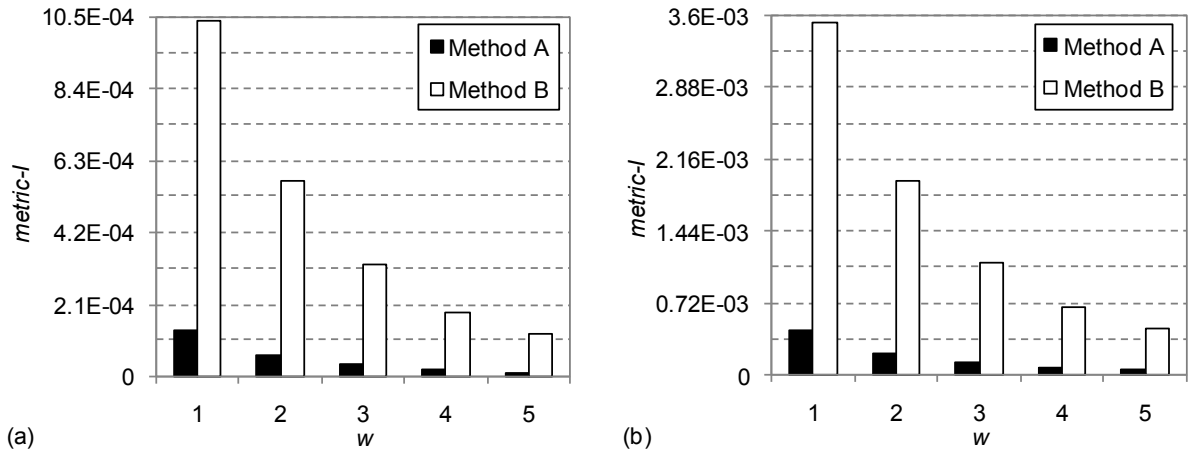


Fig. 6.19. *metric-I* for platform state XIV based on the theoretical (a) and practical (b) complexities.

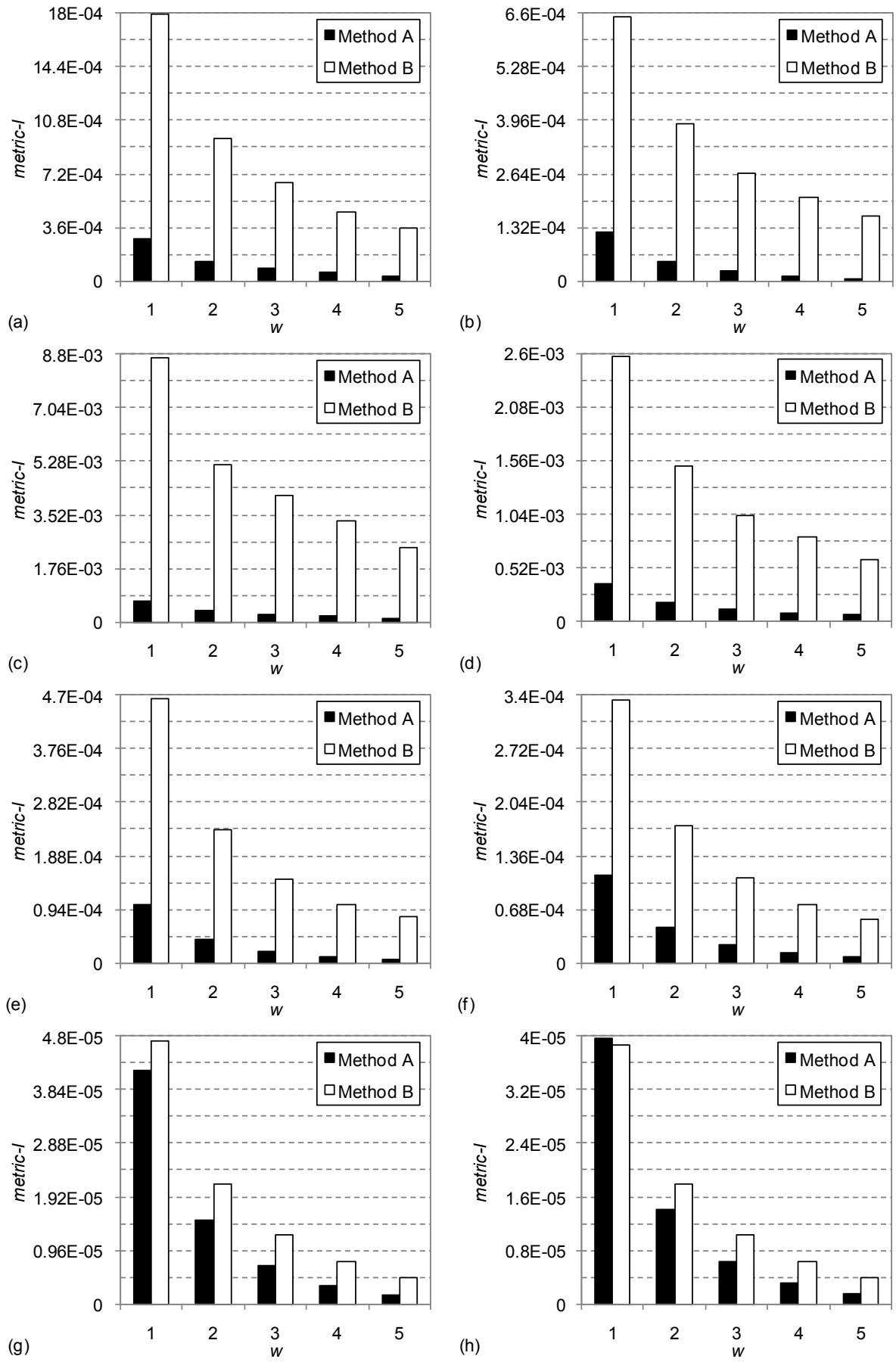


Fig. 6.20. *metric-I* for platform states XV-XXII (a)-(h) based on the practical complexities.

Table 6.7 Mean number of iterations (*m-iter*) of *method B*.

w \ *D*	XIV	XV	XVI	XVII	XVIII	XIX	XX	XXI	XXII
1	1.069	1.127	1.284	1.040	1.095	1.356	1.394	2.360	2.518
2	1.037	1.067	1.168	1.019	1.048	1.226	1.246	1.985	2.120
3	1.020	1.033	1.099	1.008	1.024	1.139	1.154	1.693	1.802
4	1.011	1.017	1.057	1.004	1.012	1.083	1.094	1.478	1.560
5	1.006	1.008	1.033	1.002	1.006	1.046	1.052	1.326	1.391

Fig. 6.20g and h show relatively small differences between *method A* and *method B* regarding *metric-I*. The relatively high numbers of infeasible reconfigurations for platform states XXI and XXII (Fig. 6.15) and the correspondingly high *m-iter* (Table 6.7) explain this. More sophisticated *q*-selection algorithms should be considered for cases like these. A simple approach would be limiting the set of *q* instances (for certain SDR applications or platforms).

6.5 Summary

This chapter has analyzed three major computing resource management issues. First we have demonstrated the relevance of the premapping order and the facilities of our framework for applying different reordering approaches. We have introduced the *c*- and *b*-ordering algorithms, which reorder SDR functions from high to low processing and from high to low bandwidth demands. We have compared the mapping results and concluded that the selection of an appropriate mapping order depends on many problem parameters. Based on several simulation results we could, though, derive some general rules, which may be valid for other reconfiguration scenarios as well. These rules should, however, be refined, taking into account additional platform and application parameters.

A better ordering approach would probably be a mix of the *c*- and *b*-ordering as a function of the application characteristics and the platform features. A set of thresholds accounting for several modeling parameters may also be meaningful to approach the multidimensional mapping problem. Dynamic reordering during the t_w-mapping process as well as other (preprocessing) techniques may also be practical. These options require more profound analyses, which are scheduled for future research.

The second analysis has studied the implications of the hardware architecture on the application mapping. We have observed that the interprocessor communication network and the distribution of computing resources affect the mapping results. More precisely, the availability and distribution of computing resources significantly condition the application mapping. The modeling parameters of Chapter 3 have facilitated the analysis of the results here. They may, moreover, be useful for predicting the outcomes of similar reconfiguration situations and making appropriate computing resource management decisions. More theoretical work is, however, necessary for making reliable and widely applicable predictions and decisions.

We have, finally, evaluated the importance of the t_w-mapping parameter *w* and the cost function parameter *q*. The simulation results have shown that both parameters are very relevant. A metric that relates performance to complexity has demonstrated that searching for a suitable *q* is more efficient than augmenting *w*. Deriving more sophisticated *q*-selection algorithms for our two-term cost function proposal as well as for other multi-objective cost functions remains for future studies.

We conclude that our SDR computing resource management framework is flexible. Its modular design facilitates a situation-dependent computing resource management. We have indicated how to adjust the computing resource management parameters to different reconfiguration scenarios. Nevertheless, other problems may behave differently and may require additional analyses for finding the most appropriate parameter sets.

7

Computing Resource Management in Cognitive Radio

7.1 Introduction

So far we have argued for computing resource management in SDR. Cognitive radio, the next step toward flexible wireless communications, is generally considered as *the* application of SDR: SDR allows for flexible reconfigurations of radio equipment and facilitates a situation dependent use of resources. Cognitive radio can take advantage of these reconfiguration capabilities to achieve better resource efficiencies by means of intelligent allocations. We, therefore, assume a cognitive radio system that is based on SDR technology.

As we have argued throughout this document, resources for SDR communications include, apart from the radio resources, also the computing resources. Computing resources are essential for running (software-defined) signal processing algorithms, among others. Computing resource management is, hence, important in SDR and, consequently, in SDR-based cognitive radio contexts as well. This chapter examines the possibilities of computing resource management in cognitive radio and extends the scope of our SDR computing resource management framework of Chapters 3 and 4. We review some related work (Section 7.2) before introducing the *cognitive computing resource management* (Section 7.3) and indicating its suitability and relevance in cognitive radio scenarios (Section 7.4).

7.2 Related Work

Cognitive radio is primarily associated with RRM and dynamic spectrum access (Section 1.4.1B)). These are timely topics, but are not the only issues of today's radio communications research. Consequently, the scope of cognitive radio cannot be limited to radio resource or spectrum management. This section points out some cognitive radio research incentives regarding computing resources management as well as a few general contributions.

Mitola's dissertation formally identifies *radio resource management*, *network management*, *service delivery*, and *download certification* as the four areas of influence of cognitive radio [115]. It, moreover, introduces the notion of self-awareness, which refers to a computational model the system has of itself. The Radio Knowledge Representation Language (RKRL), particularly, provides the means for representing the types of processors, their number, and their processing capacities. It specifies mechanisms for characterizing the time that is allocated to programs, modules, and hardware execution. This information is used for controlling the execution time of the cognition cycle (Fig. 1.4a) and for detecting software-software and software-hardware incompatibilities, rather than for managing the available computing resources.

Reference [114] essentially summarizes [115]. The following two citations from [114] are especially interesting here:

- *A GSM radio's equalizer taps reflect the channel multipath structure. A network might want to ask a handset "How many distinguishable multipath components are you seeing?" Knowledge of the internal states of the equalizer could be useful because in some reception areas, there may be little or no multipath and 20 dB of extra Signal-to-Noise Ratio (SNR). Software radio processing capacity is wasted running a computationally intensive equalizer algorithm when no equalizer is necessary. That processing capacity could be diverted to better use, or part of the processor might be put to sleep, saving battery life. In addition, the radio and network could agree to put data bits in the superfluous embedded training sequence, enhancing the payload data rate accordingly.*
- *The Act step consists of allocating computational and radio resources to subordinate (conventional radio) software and initiating tasks for specified amounts of time.*

The first quotation basically says that adjusting a GSM equalizer as a function of the channel characteristics for using just the necessary processing power could save processing resources and enhance the battery life. Saving battery is of high importance in the era of power-hungry multimedia applications and lightweight portable devices and motivates for computing resources management in modern wireless communications. The second statement indicates the potential of cognitive radio for an intelligent allocation of radio and computing resources. This potential is, however, not further explored in the paper.

Haykin [116] provides a tutorial-like introduction to cognitive radio. He first resumes the main characteristics of cognitive radio and the related RRM terminology. He then introduces the space-time processing considerations for the radio-scene analysis (I) and indicates the possibilities and difficulties of obtaining the channel-state estimation (II) using a predictive modeling. The paper presents different strategies concerning the transmit-power control (IIIa), including game theoretic and water filling approaches, and discusses some dynamic spectrum management issues (IIIb). It finally provides an outlook on some relevant research topics for the evolution of cognitive radios. Therein, the computing requirements are discussed in the sense of supporting the threes cognitive tasks I-III. This chapter, on the other hand, argues for computing resource management being another cognitive task.

Reference [117] summarizes a set of platform parameters that characterize the capabilities of SDR platforms and, specifically, the capabilities of dedicated components for the RF and IF processing and the amount of computing resources for the digital baseband processing. Computing resource management, which could take advantage of this hardware information, is, however, not mentioned.

Apart from spectrum management issues, [119] mentions the *awareness of processing capabilities for the partitioning or the scheduling of processes* as a technology centric property of cognitive radios. This concept is, however, not further addressed in the paper, which presents cognitive radio as an evolution of SDR, hence considering SDR as an enabling technology for the intelligent spectrum management. Other cognitive radio contributions introduce concepts, algorithms, and implementations related to dynamic spectrum access, a topic that is beyond the scope of this dissertation.

7.3 Cognitive Computing Resource Management

This section introduces the cognitive computing resource management concept. We specify the extended computing environment (Section 7.3.1), present our vision of a future cognitive radio system (Section 7.3.2), introduce the cognitive computing cycle (Section 7.3.3), and discuss a suitable computing system modeling (Section 7.3.4).

7.3.1 Extended Computing Environment

The SDR computing environment of Section 2.3.1 consists of the platform and application environments. These environments specify the computing resources and requirements. Here we consider a more general approach: The platform resources of the *extended computing environment* encompass all computing and computing-related resources of SDR equipment, ranging from the analog RF part to the digital data processing resources for higher OSI layer implementations and including software mapping and control mechanisms. The platform resources thus include

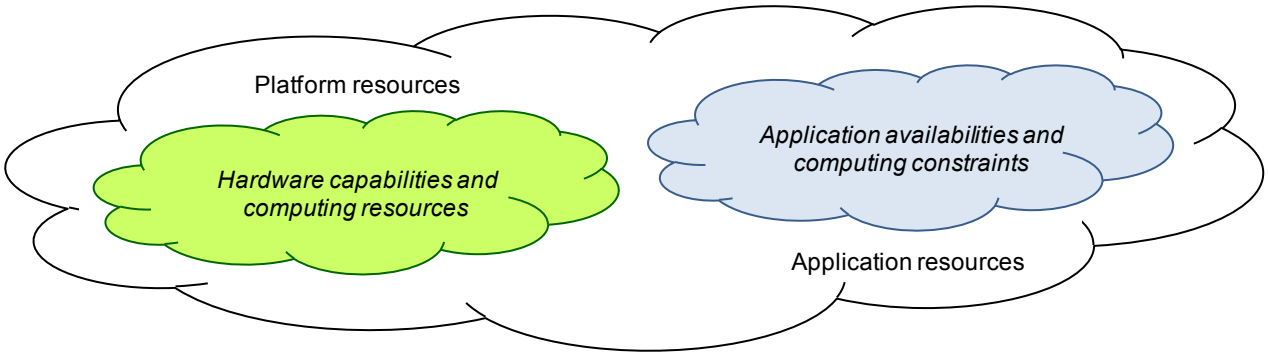


Fig. 7.1. Extended computing environment.

- the analog circuitry or resources (for the RF and IF signal processing),
- digital processing powers and interprocessor bandwidths,
- memory resources,
- energy resources,
- software mapping mechanisms, and
- execution control and management entities.

The software availabilities and their constraints correspondingly define the application resources. The extended computing environment then also captures the available software downloads─SDR applications and functions─and their computing requirements. Fig. 7.1 illustrates this. It indicates the *computing environment* that this chapter assumes.

Some SDR platform resources offer limited flexibility or no flexibility at all, such as a single RF front end, and require only minimal management. Considering different types of platform resources, however, ensures a general approach, which may be valid for SDRs with different degrees of flexibility, from parameter-controlled [16] to ideal software radios [4].

The evolution of hardware capabilities facilitates the introduction of more sophisticated communications technologies (application resources) and services. Even though the processing power may once grow faster than its requirement, related issues, such as energy consumption, will not become obsolete soon. Hence, it is and will remain necessary to optimize the usage of the limited platform resources. As communications evolve from SDR to cognitive radio [119], so must also the computing resource management.

7.3.2 Extended Cognitive Radio System

We argue for an extension of the cognitive radio concept and suggest a cognitive radio system that also accounts for the computing environment and its implications on wireless communications. Such a system would behave according to the available platform and application resources. Despite the additional constraints, the computing constraints, the system could eventually take advantage of the information about the computing environment. We analyze this throughout the chapter.

This section presents our view of a future cognitive radio system. We introduce a high-level architecture that implicitly addresses the computing resource management. We, therefore, assume the layered software radio architecture of Fig. 1.2 and build the cognitive functionalities around this architecture. Fig. 7.2 illustrates this.

The *intelligent subsystem* at the top of Fig. 7.2 and the two interfaces on the left and right provide the cognitive functionalities. These interfaces offer *sensing and monitoring* facilities, on the one hand, and *adaptive mechanisms*, on the other.

The layered software radio architecture was introduced by Mitola [5] and discussed in Section 1.2. It distinguishes between the *communications services*, the *radio applications*, the *radio infrastructure*, and the *hardware platform*. Each of these logical layers is, to a certain degree, individually adaptable. Communications services can be dynamically adapted to the momentary radio conditions (interference, propagation loss, and so forth). If, for instance, the radio channel suddenly worsens, a real-time video service may be adapted from high resolution to low resolution, or even exchanged for a pure audio service if necessary and accepted by the user.

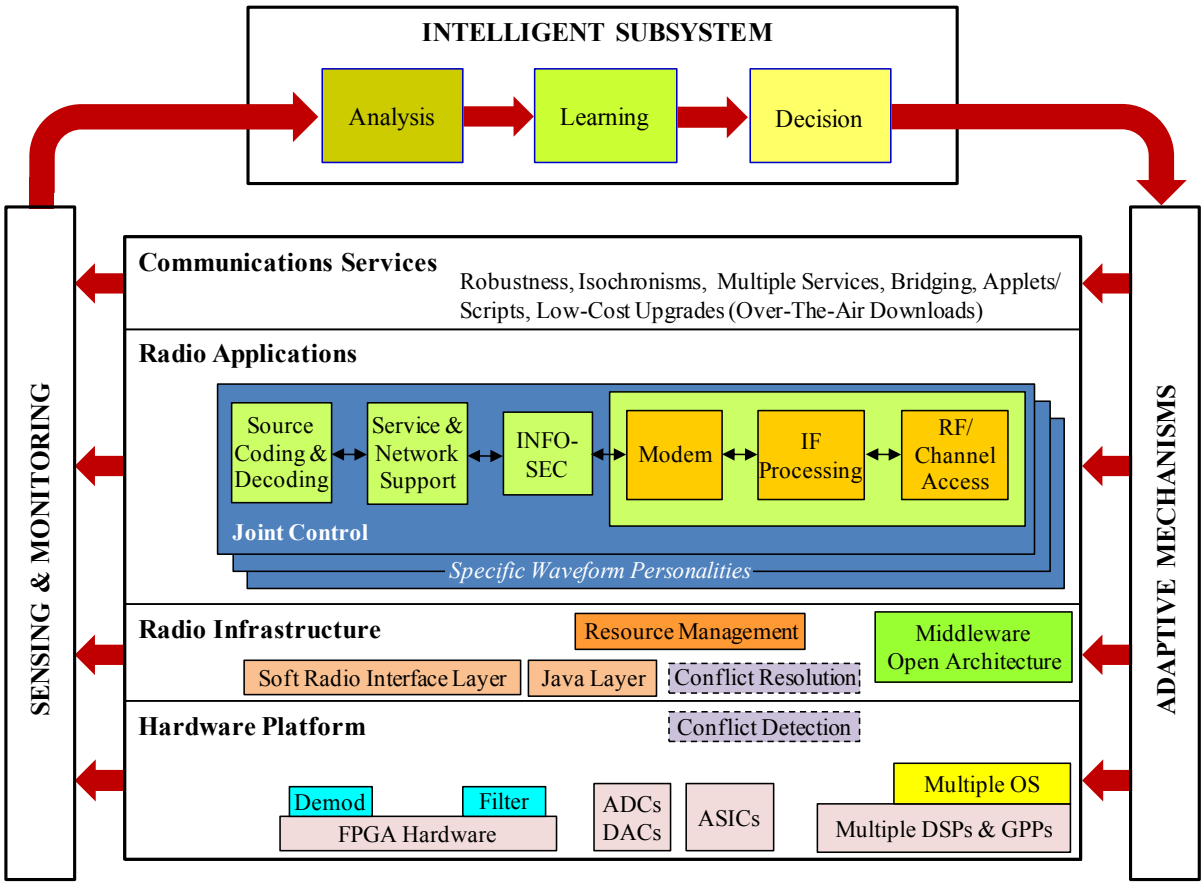


Fig. 7.2. Functional blocks of the extended cognitive radio system.

Communications services are facilitated through radio applications. Radio applications represent the (digital) signal processing chains of radio standards. They define the transmission and reception modes for correct service delivery over the wireless link. The wireless channel conditions the applicability of radio applications: Following the previous example, certain channel states may require exchanging a high-resolution video communications service for a low-resolution video or audio transmission and reception and may require using another radio application. Also, application's modules, such as the coding or error correction mechanisms, may be individual adaptable as a function of the signal-to-noise ratio (SNR) (Section 7.2).

The resource management block of the radio infrastructure is, among others, responsible for mapping radio applications to the available hardware platform. It generally tries to optimize the usage of the available computing resources and, thus, aims at computing efficient implementations of radio applications. The hardware platform or its current state, though, constrains the applicability of radio applications and, eventually, the delivery of communications services. The radio infrastructure needs to deal with these issues and support the proper selection of radio applications.

The *sensing and monitoring* interface scans the different layers of the software radio architecture, processes their states and provides any changes to the intelligent subsystem. This process needs to run continuously, although its periodicity may vary. The sensing and monitoring cycle could be dynamically adjusted by the intelligent subsystem as a function of several internal and external system parameters.

The *intelligent subsystem* processes the environmental information provided by the sensing and monitoring interface. The first phase consists of analyzing the system's current conditions. In the next phase, these results would be compared with previous states to evaluate whether the previous decisions have positively or negatively affected the overall system performance. This way the system could learn how to correctly face repeating situations and would have a better base for solving new problems. The decision phase finally follows a set of rules for deriving the most appropriate decisions as a function of the environmental analysis and the acquired knowledge.

The *adaptive mechanisms* or information exchange interface between the intelligent subsystem and the layered software radio architecture provides two functionalities: adapting the decisions of the intelligent

subsystem to the particular software radio architecture implementation and triggering the different internal management layers for executing the corresponding actions. The second task is important because some decisions may affect only one system layer, whereas others may require adjustments of various layers.

The above discussion indicates several interdependencies between communications services, radio applications, and computing resources. If we assume that future cognitive radio systems will be build around the above or similar software radio architectures, cognitive computing resource management will become an essential part of the overall resource management in cognitive radio. We, therefore, introduce the cognitive computing cycle.

7.3.3 Cognitive Computing Cycle

Mitola [114] presented the *cognition cycle*, which consists of the *observe*, *orient*, *plan*, *decide*, and *act* phases (Fig. 1.4a). These phases coordinately process the information from the outside world, the nonstationary radio environment, for responding with the appropriate reactions [115]. On the basis of this cycle, Haykin [116] derived the *basic cognitive cycle*, which features the *radio scene analysis*, the *channel-state estimation and predictive modeling*, and the *transmit-power control and spectrum management* (Fig. 1.4b).

The SDR computing environment is nonstationary as well: SDR platforms will be upgraded and new ones introduced, SDR applications and functions will be updated and new ones created, and computing resources will be dynamically allocated and released. Fig. 7.3, therefore, introduces the *cognitive computing cycle*.

The *computing scene analysis* entity of the cognitive computing cycle continuously scans the computing environment. It collects the hardware and software state information of radio equipment (mobile terminals and network elements) and software repositories and provides any changes to the *computing resource management* (CRM) entity. The CRM entity is the brain of the system: it reacts with the most suitable actions as a function of its inputs and the available knowledge. This knowledge is accumulated during the learning process of the system, which observes the effects of its decisions (*learning-by-observing*). Hence, the cognitive computing cycle runs continuously.

A cognitive radio system should be able to select the appropriate radio equipment for reconfiguration. The limited computing resources may then be more efficiently allocated, either locally or globally, increasing the system's computing efficiency. This, in turn, facilitates the distribution of radio resources: If, for example, the radio resources of RAT *A* are overused, while those of RAT *B* are not (radio scene analysis) and if the network infrastructure and mobile terminals facilitate a switch to RAT *B* (computing scene analysis), the action could be reconfiguring some SDR-MTs to use RAT *B* instead of RAT *A*. Such a cooperative radio and computing resource management will be discussed in Chapter 8.

The computing cycle of Fig. 7.3 is valid SDR-MTs as well as for SDR network elements. Although the computing resources of the network infrastructure are less limited, an intelligent management may

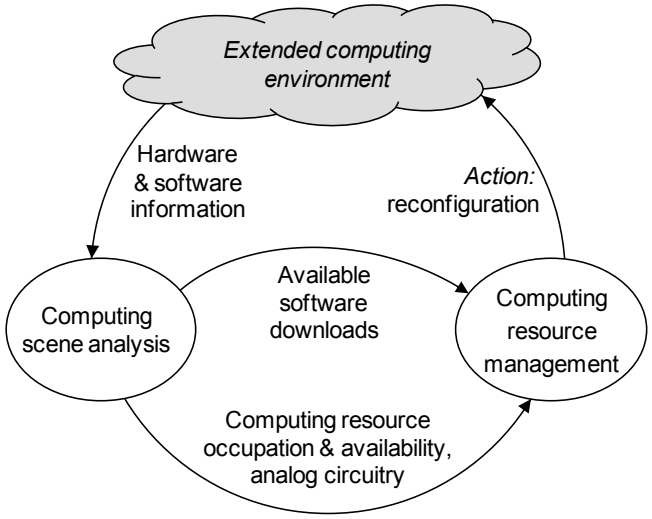


Fig. 7.3. The cognitive computing cycle.

reduce the operational costs. Radio operators can then expect a more efficient usage of computing resources and focus on offering a higher variety of transmission modes, services, and QoS levels.

A reconfiguration of an SDR-MT, either partial or total, may require an end-to-end coordination of several communication layers. The reconfiguration of an SDR-MT may, particularly, invoke the reconfiguration of some network elements and vice versa. An example would be dynamically adjusting the downlink transmission data rate as a function of the terminal's momentary processing capabilities. Certain computing states, such as energy shortage, could then be a reason for reconfiguring SDR-MTs, network elements, or both.

A cognitive computing resource management would also facilitate the anywhere, anytime, and anyhow wireless access, where the wireless subscriber could be unaware of the RATs that provide the services. Therefore, the globally distributed cognitive radio systems would need to be coordinated for cooperatively managing the distributed platform and application resources.

7.3.4 Computing System Modeling

The cognitive computing resource management implies the continuous monitoring of computing resources and requirements (computing scene analysis). This requires a computing system modeling that can dynamically be updated and modified. We have presented an SDR computing system modeling in Chapter 3. It consists of two general modeling templates $(\boldsymbol{R}_{\mathrm{P}}{}^{t'})^{\boldsymbol{D}}$ (3.1) and $(\boldsymbol{R}_{\mathrm{A}}{}^{t''})^{\boldsymbol{d}}$ (3.8) for modeling the platform and the application features, where t' and t'' (t', $t'' \in \mathbb{N}$) index the platform and application features of SDR platform D and SDR application d, respectively. The two special modeling templates $(\boldsymbol{R}^{t})^{\boldsymbol{D}} \subseteq (\boldsymbol{R}_{\mathrm{P}}{}^{t'})^{\boldsymbol{D}}$ (3.2) and $(\boldsymbol{r}^{t})^{\boldsymbol{d}} \subseteq (\boldsymbol{R}_{\mathrm{A}}{}^{t''})^{\boldsymbol{d}}$ (3.9) capture the computing resources and requirements for any computing resource type t ($t \in \mathbb{N}$) (Section 3.3).

The above templates can be instantiated as often as necessary for capturing all relevant platform and application features, including computing resources and requirements. The modeling approach of Chapter 3 thus permits a dynamic and flexible resource monitoring (platforms and applications), because modeling matrices can be easily added, updated, redimensioned, discarded, or substituted. The cognitive computing cycle can then continuously monitor and instantly update the information about computing resources and requirements, facilitating the cognitive computing resource management.

Several modeling instances, including different models for capturing the processing and interprocessor bandwidth resources and requirements, have already been exposed in Chapter 3. Processing and bandwidth resources are the principal computing resources for SDR communications. Additional resources are, though, required. To illustrate the suitability of the modeling approach for tracking the states of those resources and requirements that present (significant) computing constraints, we introduce additional platform models as instances of $(\boldsymbol{R}_{\mathrm{P}}{}^{t'})^{\boldsymbol{D}}$ and $(\boldsymbol{R}^{t})^{\boldsymbol{D}}$ and the corresponding application models as instance of $(\boldsymbol{R}_{\mathrm{A}}{}^{t''})^{\boldsymbol{d}}$ and $(\boldsymbol{r}^{t})^{\boldsymbol{d}}$. We, therefore, continue the numbering of platform features (t'), application features (t''), and computing resources (t) from Section 3.3.

A) Additional Platform Models

SDR platform D may have a global memory for storing the downloaded SDR application of the new radio mode. The application mapping process then distributes the SDR functions among the processors' local memories. The memory availability for program and data can then be modeled as

$$(\boldsymbol{R}_{\mathrm{P}}{}^{5})^{\boldsymbol{D}} = (\boldsymbol{R}^{4})^{\boldsymbol{D}} = \boldsymbol{MEM}^{\boldsymbol{D}} = ((MEM_1)^D, (MEM_2)^D, \ldots, (MEM_{N(D)})^D, (MEM_{N(D)+1})^D) \text{ MB}, \tag{7.1}$$

where $(MEM_1)^D$ through $(MEM_{N(D)})^D$ represent the processors' local and $(MEM_{N(D)+1})^D$ the platform's global memory resources in megabytes (MB). The energy resources can be modeled as

$$(R_{\mathrm{P}}{}^{6})^D = (R^5)^D = E^D \text{ mWPS}, \tag{7.2}$$

if we assume that a single energy resource of E^D in milliwatts per second (mWPS) is shared among all computing resources of SDR platform D or as

$$(\boldsymbol{R}_{\mathrm{P}}{}^{7})^{\boldsymbol{D}} = (\boldsymbol{R}^{6})^{\boldsymbol{D}} = \boldsymbol{E}^{\boldsymbol{D}} = ((E_1)^D, (E_2)^D, \ldots, (E_{N(D)})^D) \text{ mWPS}, \tag{7.3}$$

if energy resources are manageable on processor basis.

B) Additional Application Models

SDR application d will require a certain memory space for its execution. Therefore,

$$(\boldsymbol{R}_{\mathrm{A}}^{6})^{d} = (\boldsymbol{r}^{4})^{d} = \boldsymbol{mem}^{d} = ((mem_1)^{d}, (mem_2)^{d}, \ldots, (mem_{M(d)})^{d}, (mem_{M(d)+1})^{d}) \text{ MB} \quad (7.4)$$

informs about the memory requirements of application d, where $(mem_1)^d$ through $(mem_{M(d)})^d$ represent the SDR functions' program and data memory requirements and $(mem_{M(d)+1})^d$ the memory demand of the application's software download file.

The SDR applications' energy demands also need to be met. They may be given on SDR application basis,

$$(R_{\mathrm{A}}^{7})^{d} = (\boldsymbol{r}^{5})^{d} = e^{d} \text{ mWPS}, \quad (7.5)$$

or on SDR function basis,

$$(\boldsymbol{R}_{\mathrm{A}}^{8})^{d} = (\boldsymbol{r}^{6})^{d} = \boldsymbol{e}^{d} = ((e_1)^{d}, (e_2)^{d}, \ldots, (e_{M(d)})^{d}) \text{ mWPS}. \quad (7.6)$$

These three application models are symmetric to the platform models (7.1), (7.2), and (7.3). The computing resources and requirements generally correspond, although the platform and application features may not. (There may be features that are specific to either platforms or applications.) This facilitates allocating computing resources of a certain type t to the corresponding computing requirements, as the previous chapters have demonstrated for processing and bandwidth resources and requirements.

7.4 Proof of Concept

It is difficult to fairly compare our proposal against typical cognitive radio system implementations that assume no restriction on the availability of computing resources. Such an assumption may be appropriate when dealing with multimode radio terminals, but is not suitable for SDR-based cognitive radio systems, where computing resources are flexibly assigned and released. Since considering these systems, we evaluate our cognitive computing resource management proposal in scenarios where computing resources are limited (SDR scenarios). We discuss two basic scenarios (Section 7.4.1) and analyze simulation results (Section 7.4.2) for justifying our approach and indicating its potentials.

7.4.1 Basic Scenarios

A) Scenario I

We first consider a basic scenario where SDR-MTs of classes 1, 2, and 3 are reconfigured to different modes. The three terminal classes are distinguished by their computing capacities, which are specified in computing units (CU). This metric abstracts a certain computing resource, such as processing power, interprocessor bandwidth, memory availability, or a combination of resources.

The SDR-MTs are dynamically reconfigured from some radio transmission mode to mode A, B, or C. Mode switches during user sessions are assumed here so that an unsuccessful reconfiguration would correspond to a lost user session. The amounts of required CU characterize the three radio modes. Fig. 7.4a indicates the computing requirements and capacities of the radio modes and terminal classes, respectively. It presents the compatibility matrix between the modes and classes, where '+' indicates compatibility and '–' incompatibility. We assume that modes A and B facilitate delivering equivalent services and QoSs, whereas mode C cannot provide the same QoS as the other two modes for all services.

The baseline algorithm, Alg-0, tries to reconfigure an SDR-MT to the desired mode from the RRM and QoS perspective. Alg-1, on the other hand, also processes the information about the computing environment, captured by the compatibility matrix. The scenario assumes that enough radio resources are available for reconfiguring each terminal to any of the three modes. Later on we will justify this assumption.

In cases where a user desires a higher QoS than mode C can provide, the SDR-MT needs to be reconfigured to either mode A or B. Alg-1 chooses mode A, mode B, or no reconfiguration as a function of the terminal's computing resources: If the terminal (momentarily) neither supports mode A nor mode B, the service is maintained at the lower QoS of mode C. Alg-0, on the other hand, initiates the reconfiguration in any case.

Mode	Class 1 100 CU	Class 2 80 CU	Class 3 60 CU
A (90 CU)	+	–	–
B (70 CU)	+	+	–
C (50 CU)	+	+	+

(a)

	Uninterrupted sessions	Lost sessions	QoS degradations
Alg-0	66.6 %	33.3 %	0 %
Alg-1	100 %	0 %	22.2 %

(b)

Fig. 7.4. Compatibility matrix (a) and statistical results (b) for scenario I.

Mode	Class 1 100 CU	Class 2 80 CU	Class 3 60 CU
A (90 CU)	–	–	–
B (70 CU)	+	–	–
C (50 CU)	+	+	+

(a)

	Uninterrupted sessions	Lost sessions	QoS degradations
Alg-0	44.4 %	55.5 %	0 %
Alg-1	All but 2	2	~44.4 %

(b)

Fig. 7.5. Compatibility matrix (a) and statistical results (b) for scenario II.

Fig. 7.4b shows the statistically expected results. Despite the basic scenario, these results indicate some advantages of a cognitive radio system that implements our cognitive computing resource management: Being constantly aware of the terminals computing capabilities (Alg-1), it can avoid infeasible reconfiguration requests while sacrificing QoS only for the sake of connectivity.

A) Scenario II

Scenario II demonstrates the most basic way of knowledge acquisition. It accounts for the difficulty of allocating distributing computing resources. The corresponding compatibility matrix (Fig. 7.5a) indicates that theoretically enough computing resources do not necessarily guarantee a feasible mode. Alg-1 needs to acquire this knowledge, starting from its initial knowledge of Fig. 7.4a. It can obtain it through experience: By making a wrong decision and losing a user session, the system learns to avoid this decision, excluding this class-mode pair. This way the system eventually gains the necessary knowledge for avoiding infeasible reconfiguration requests.

The statistical results (Fig. 7.5b) show that Alg-1 loses sessions in this scenario, but only during its learning phase. The percentage of QoS degradations of Alg-1 is, again, lower than the percentage of lost sessions of Alg-0. This is so because QoS degradations are only accepted if necessary for maintaining the connection.

The B3G context is characterized by several overlapping RATs, in this case three. The limited availability of radio resources in practice will usually not allow such a flexible selection of modes. But even in the extreme case with only one momentarily available mode, the cognitive computing resource management would make sense as a support for the redistribution of radio resources. If, for example, mode B, which is interesting because of its low computing resource demands and high QoS capabilities, is momentarily overused, the system may initiate the reconfiguration of some SDR-MTs that run in mode B to use mode A or C, whichever available, according to the terminals' computing capabilities and the users' QoS demands.

7.4.2 Simulations

We consider a typical case study and a simple simulation setup for evaluating two basic CRM algorithms or policies. Without loss of generality, we focus on the reconfiguration of SDR-MTs.

A) Case Study

The spectrum scanning at a certain time and area with 2G and 3G coverage shows that the number of GPRS sessions is elevated. The cognitive radio system thus decides to reconfigure some SDR-MTs that run in the GPRS mode to operate in another mode. This way the network capacity could be increased and more users satisfied. We assume that the UMTS radio resources are underused. A reconfiguration of some SDR-MTs that run in the GPRS mode to access the UMTS RAT would then balance the traffic loads between the two wireless standards so that penetrating 2G mobile terminals, as opposed to SDR-MTs, could maintain or initiate their GPRS sessions.

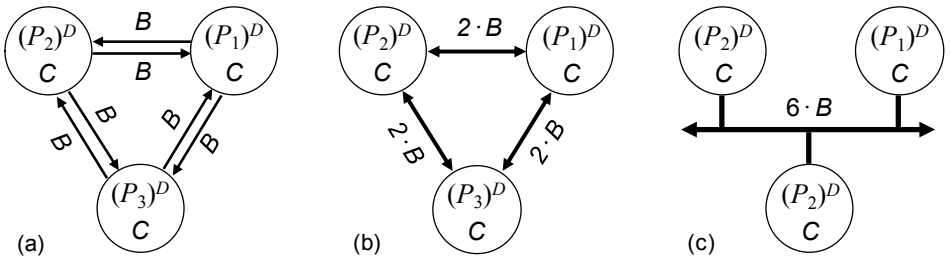


Fig. 7.6. SDR platform types XXIII-XXV (a)-(c).

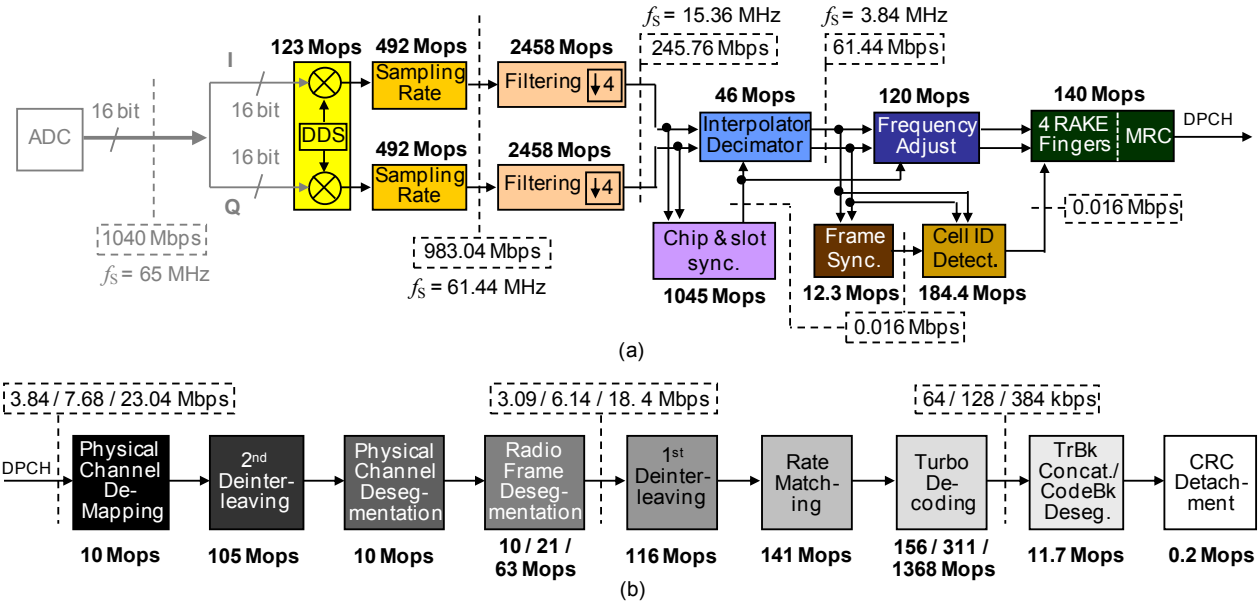


Fig. 7.7. SDR applications: chip- (a) and bit-rate (b) computing requirements of the 64, 128, and 384 kbps UMTS downlink receivers.

Table 7.1 Some implementation details.

SDR function	Implementation details
Digital down converter	@ 65 MHz
Sampling rate adjustment	output @ 61.44 MHz = 16 · chip rate
Filter	channelization and matched filters
Chip & slot synchronization	2 matched hierarchical real correlators, each @ 4 · 3.84 MHz [107], [109]
Frame synchronization	32 256-chip real correlators @ 1.5 kHz [107], [109]
Cell ID detection	descrambling and correlation [107]
RAKE receiver	4 fingers, SF = 4 & 8 [107], [110]
Turbo decoder	4 iterations [107], [108]

Our simulation time span is specified by 15 000 SDR-MTs of different computing architectures and resources. All of these terminals initially run in the GPRS mode. The objective is to reconfigure as many SDR-MTs as possible to access the WCDMA air interface. We assume that additional applications may concurrently run on an SDR-MT. Hence, part of a platform's total computing resources may not be available for implementing the new radio mode.

B) SDR Platforms

Fig. 7.6 shows three types of SDR-MTs. Fig. 7.6a illustrates a dedicated FD communication network, where the three processors are completely interconnected through dedicated and unidirectional communi-

cation lines. The HD architecture of platform XXIV permits a more flexible bandwidth assignment between any processor pair. The bus system of Fig. 7.6c permits a fully flexible bandwidth distribution. The mean link flexibilities (Table 3.4) of the three architectures are $(LF_M)^{XXIII} = 1/6$, $(LF_M)^{XXIV} = 1/3$, and $(LF_M)^{XXV} = 1$. We assume that each platform runs the P-HAL-OE (Section 2.2.3) and that it features the t_1-mapping algorithm with cost function (4.9) (Sections 4.2.1 and 4.3.2).

These platform types represent a small though representative excerpt of the numerous possible SDR platform architectures. $(C_T)^D = 3 \cdot C$ [MOPS] and $(B_T)^D = 6 \cdot B$ [Mbps] quantify a platform's total processing and bandwidth resources for executing the SDR applications of Section 7.4.2C).

This case study addresses 5000 SDR-MTs of each type. Their reconfigurable computing resources C and B are simulated as independent random variables, which are uniformly distributed in {2700, 2750, …, 4200} MOPS and {200, 250…, 1500} Mbps, respectively. This way we simulate SDR platforms of different architectures and resources.

C) SDR Applications

The SDR applications are three partial UMTS downlink receiver implementations, providing data rates of up to 64, 128, and 384 kbps. Each of these SDR applications comprises 20 SDR functions. Fig. 7.7 illustrates the three processing chains and their computing requirements. The computing requirements were obtained from [107], [108], [109], [110] and available implementations. An SDR function's processing requirement was calculated as the number of MACs times the required processing frequency, which is a function of the sampling rate f_S. Likewise, a bandwidth demand represents the product of the data rate and the bit precision of 16 bits per sample. Table 7.1 provides further implementation details.

Note that some SDR functions are implemented in another way than the corresponding functions of the UMTS receiver of Fig. 3.4. The chip and slot synchronization, in particular, is more efficiently implemented here, taking advantage of the hierarchical properties of the P-SCH sequence of the UMTS standard, among others [107], [109]. This indicates the flexibility of SDRs regarding the introduction of more efficient software implementations, a topic that is further discussed in Chapter 8.

The total processing requirements of the three SDR applications are approximately $(c_T)^{64kbps} = 8130$ MOPS, $(c_T)^{128kbps} = 8300$ MOPS, and $(c_T)^{384kbps} = 9400$ MOPS. The total processing requirement of the GPRS receiver is roughly an order of magnitude lower than that of an UMTS receiver [19], [141]. The GPRS receiver is thus feasibly mappable to (a single processor of) any SDR-MT. The time slot duration and the other modeling parameters of Chapter 3 are irrelevant here.

The processing and bandwidth resources of Section 7.4.2B) correspond to the available computing resources after the (projected) deallocation of the GPRS receiver functions. They are dedicated to the partial receiver implementations illustrated in Fig. 7.7. We assume that additional computing resources are available for implementing the higher OSI layers and the UMTS and GPRS uplink transmitters. We also assume that other types of computing resources, such as memory and energy, though required, do not constrain the applicability of these SDR applications.

D) Baseline and CRM Algorithms

Baseline Algorithm (Alg-0)

The baseline algorithm is not aware of the SDR-MTs' computing capabilities and, thus, initiates the reconfiguration of any candidate SDR terminal. A terminal with insufficient computing capacity for executing the UMTS signal processing chain fails in switching from GPRS to UMTS, interrupting the currently running session. The system chooses the 384 kbps UMTS realization because it can provide the highest QoS.

CRM Algorithm (Alg-1)

Alg-1 uses the supplied hardware and software information. It reconfigures an SDR-MT as a function the platform's computing resources, the applications' computing requirements, and the available reconfiguration information. This information is successively accumulated during the cognitive learning process: If a session is lost because of an unsuccessful reconfiguration intent (*infeasible mapping*, which means that the computing or timing constraints cannot be met), terminals with equivalent computing characteristics are not considered again for reconfiguration.

The system dynamically chooses the UMTS realization as a function of the total processing capacity $(C_T)^D$ of SDR-MT or platform D:

- 384 kbps UMTS if $(C_T)^D \geq (c_T)^{384kbps}$,
- 128 kbps UMTS if $(c_T)^{384kbps} > (C_T)^D \geq (c_T)^{128kbps}$,
- 64 kbps UMTS if $(c_T)^{128kbps} > (C_T)^D \geq (c_T)^{64kbps}$, and
- no reconfiguration if $(c_T)^{64kbps} > (C_T)^D$.

The granularity of $(C_T)^D$ and the total processing requirements $(c_T)^{384kbps}$, $(c_T)^{128kbps}$, and $(c_T)^{64kbps}$ lead to the three processing thresholds 9450, 8400, and 8250 MOPS. T_{384} = 9450 / 3 MOPS, T_{128} = 8400 / 3 MOPS, and T_{64} = 8250 / 3 MOPS are then the processing thresholds on processor basis.

Some services may not be maintained at the same quality with the 64 kbps or 128 kbps as opposed to the 384 kbps UMTS transceiver. We assume that the users are willing to accept a possible quality degradation for the sake of a ubiquitous wireless access. Users generally prefer a lower QoS than no service at all.

CRM Algorithm 2 (Alg-2)

Alg-2 is an extension of Alg-1. It initially chooses the same thresholds for selecting one of the three UMTS realizations. The learning process is, though, slightly different: If a reconfiguration to the 384 kbps realization fails, the next SDR-MT of the same type and equivalent computing resources will be reconfigured to the 128 kbps UMTS mode. If the 128 kbps UMTS receiver cannot be feasibly mapped, the 64 kbps implementation will be considered the next time, whereas an infeasible mapping of the 64 kbps processing chain leads to discarding equivalent terminals from being reconfigured.

E) Simulation Results

The cognitive radio system, executing one or the other CRM algorithm, sequentially evaluates the reconfiguration of SDR-MTs, processing one terminal per cognitive cycle of Fig. 7.2 and Fig. 7.3, respectively. Fig. 7.8 presents the evolution of the number of lost sessions. It shows that the curves corresponding to Alg-1 and Alg-2 saturate, reflecting the cognitive learning process: Alg-1 and Alg-2 learn which terminals can be feasibly reconfigured and which cannot. Alg-0, on the other hand, constantly loses user sessions, because not processing any information about the computing environment. Hence, Alg-1 and Alg-2 drop fewer sessions than Alg-0.

Fig. 7.9 shows the evolution of successful reconfigurations and Fig. 7.10 the operational modes of the SDR-MTs after the simulation time span. The number of SDR-MTs that remain using the GPRS mode plus those that are successfully reconfigured to any UMTS mode plus the number of lost sessions due to Fig. 7.8 is then 5000 for each platform type. These figures indicate that the cognitive computing resource management, represented through Alg-1 and Alg-2, facilitates distributing radio resource and, thus, satisfying more users in a heterogeneous radio and computing scenario.

Alg-2 loses more sessions than Alg-1 (Fig. 7.8) but also feasibly reconfigures more terminals (Fig. 7.9). The shapes of the corresponding curves in Fig. 7.8 and Fig. 7.9 indicate that Alg-1 is more suitable than Alg-2 for a low number of reconfiguration intents, whereas Alg-2 becomes the more efficient the higher the number of SDR-MTs. The algorithm selection finally depends on the system's particular objective. It is a tradeoff between losing fewer sessions and reconfiguring more terminals.

Fig. 7.11, Fig. 7.12, and Fig. 7.13 illustrate the accumulated mapping information, which is acquired during the learning process of Alg-1 and Alg-2. The complete mapping information, as shown in these figures, is only partially obtained with the 15 000 SDR-MTs. The topmost level in any figure indicates that the corresponding processing and interprocessor bandwidth resources facilitate a feasible mode switch to 384, 128, or 64 kbps UMTS. The second level indicates that the corresponding terminals can be feasibly reconfigured to the 128 or the 64 kbps realizations, whereas the first level means a maximum achievable bit rate of 64 kbps. Any *C-B* crossing at the bottom level of Fig. 7.11a, Fig. 7.12a, or Fig. 7.13a symbolizes insufficient computing resources (processing or bandwidth resources, or both) for operating in the corresponding UMTS mode. A bottom-level *C-B* crossing in Fig. 7.11b, Fig. 7.12b, or Fig. 7.13b, on the other hand, indicates that none of the three UMTS modes is feasible for the corresponding SDR-MT.

The three thresholds T_{64}, T_{128}, and T_{384} are also shown in Fig. 7.11, Fig. 7.12, and Fig. 7.13. These indicate a processor's minimum processing power C for considering the corresponding UMTS mode. For example, processing powers of $C \geq T_{384}$ indicate that Alg-1 and Alg-2 will initially choose 384 kbps UMTS as the target mode. If the reconfiguration fails, Alg-1 will not try to reconfigure equivalent terminals at all, whereas Alg-2 will consider the lower bit rate, or 128 kbps, UMTS mode (Section 7.4.2D)). The holes and steps in the figures reflect these algorithmic differences.

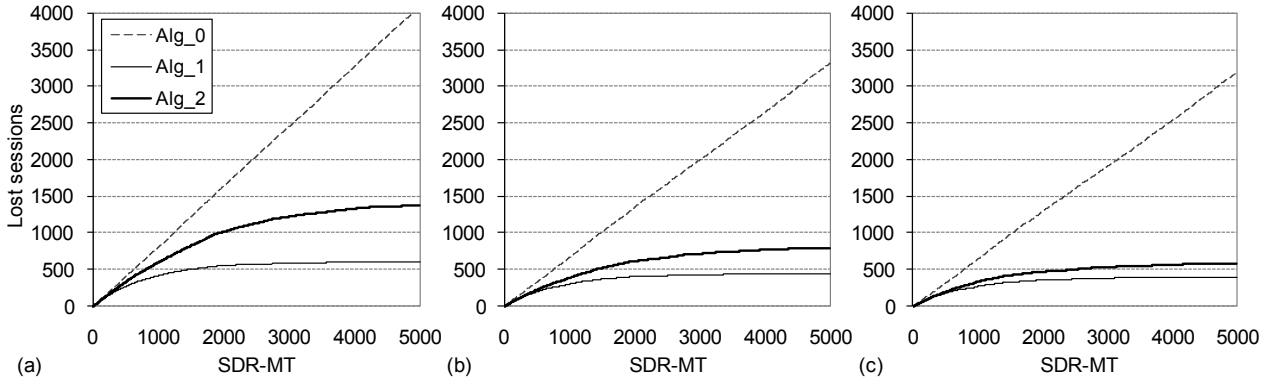


Fig. 7.8. Number of lost sessions for SDR platform types XXIII (a), XXIV (b), and XXV (c).

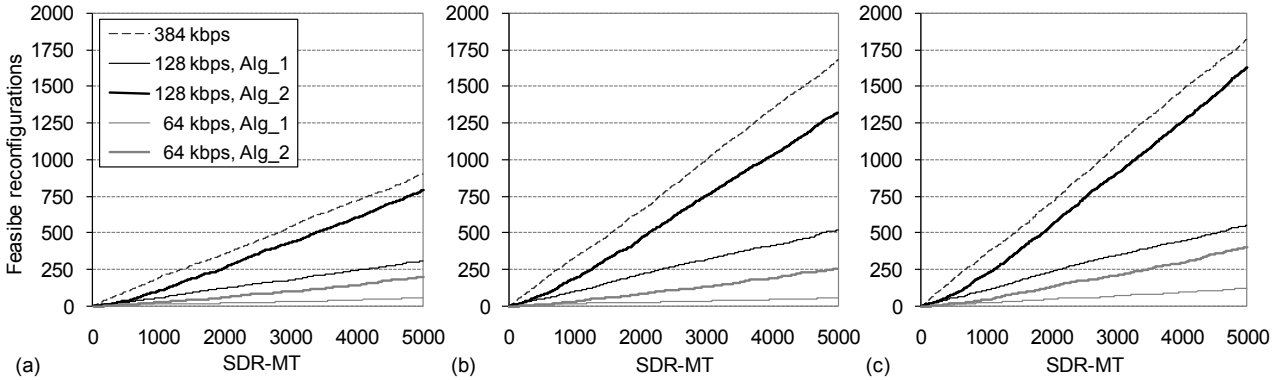


Fig. 7.9. Number of feasible reconfigurations for SDR platform types XXIII (a), XXIV (b), and XXV (c).

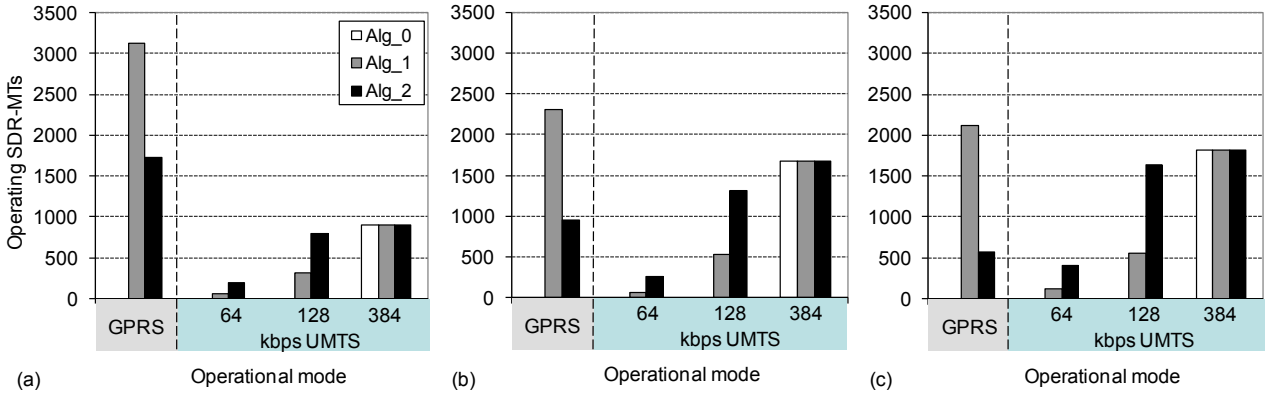


Fig. 7.10. Not reconfigured and feasibly reconfigured SDR-MTs of types XXIII (a), XXIV (b), and XXV (c).

The holes in le left subfigures and the steps not coinciding with the thresholds in the right reveal that theoretically enough processing resources do not guarantee a feasible mapping. The main reasons for this are the complexity of the mapping problem and the rather simple, though computationally practical, t_1-mapping heuristic (Chapter 4). The six subfigures of Fig. 7.11, Fig. 7.12, and Fig. 7.13 clarify the results of Fig. 7.8, Fig. 7.9, and Fig. 7.10, and vice versa. We discuss some architectural implications below.

Fig. 7.8 shows that the number of lost sessions is the highest for the SDR-MTs of type XXIII and the lowest for those of type XXV. Correspondingly, the number of feasible reconfigurations is the lowest for the terminals of type XXIII and the highest for those of type XXV (Fig. 7.9). The platforms' mean link flexibilities $(LF_M)^{XXIII} = 1/6$, $(LF_M)^{XXIV} = 1/3$, and $(LF_M)^{XXV} = 1$ explain this (Section 6.3.2).

From Fig. 7.11, Fig. 7.12, and Fig. 7.13 we, moreover, conclude that the mean link flexibility parameter provides a relative indication of a platform's necessary interprocessor bandwidth capacity $(B_T)^D$ for facilitating a feasible mapping. For $C = 3000$, for example, SDR-MTs of type XXV need merely $6 \cdot 500 = 3000$ Mbps for feasibly mapping the 128 kbps chip and bit-rate receiver processing chain, whereas those of types XXIV and XXIII require at least 3900 and 6000 Mbps, respectively, for solving the same problem

with the available computing resource management tools (t_1-mapping with cost function (4.9)). Conversely, if $(B_T)^D$ = 3900 Mbps are enough for feasibly solving a computing resource management problem at SDR-MT D of type XXIV, a feasible mapping can be predicted for the corresponding SDR-MTs of type XXV, but not for those of type XXIII.

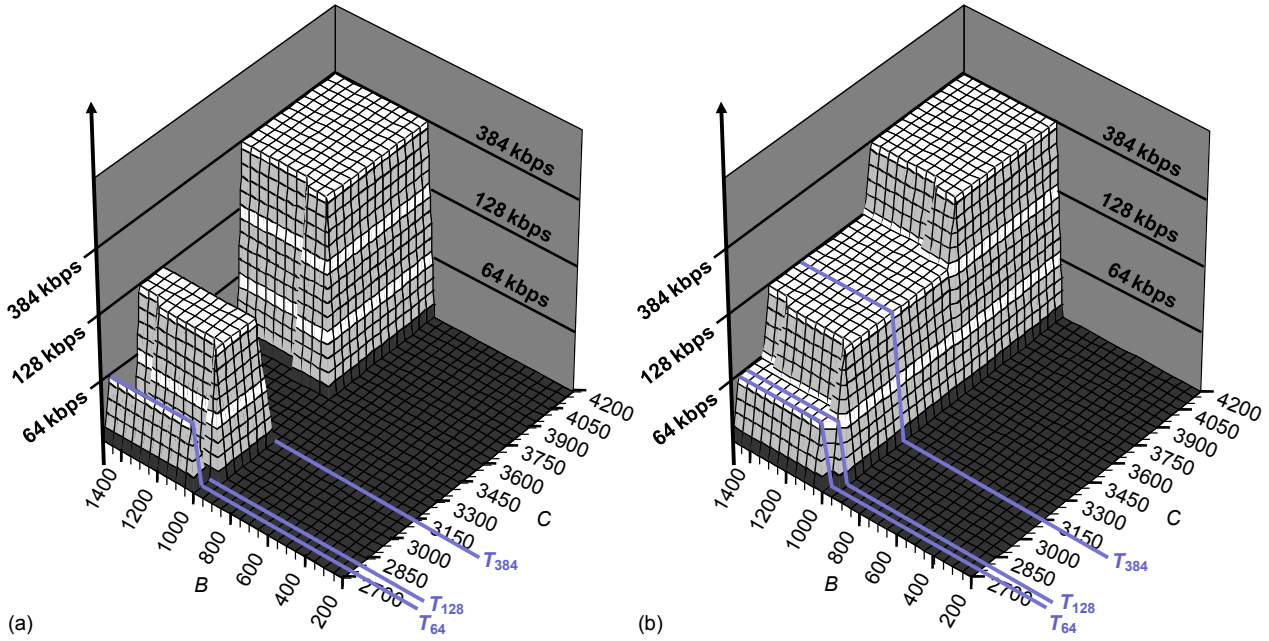


Fig. 7.11. Accumulated mapping information due to the learning process of Alg-1 (a) and Alg-2 (b) for SDR platform type XXIII.

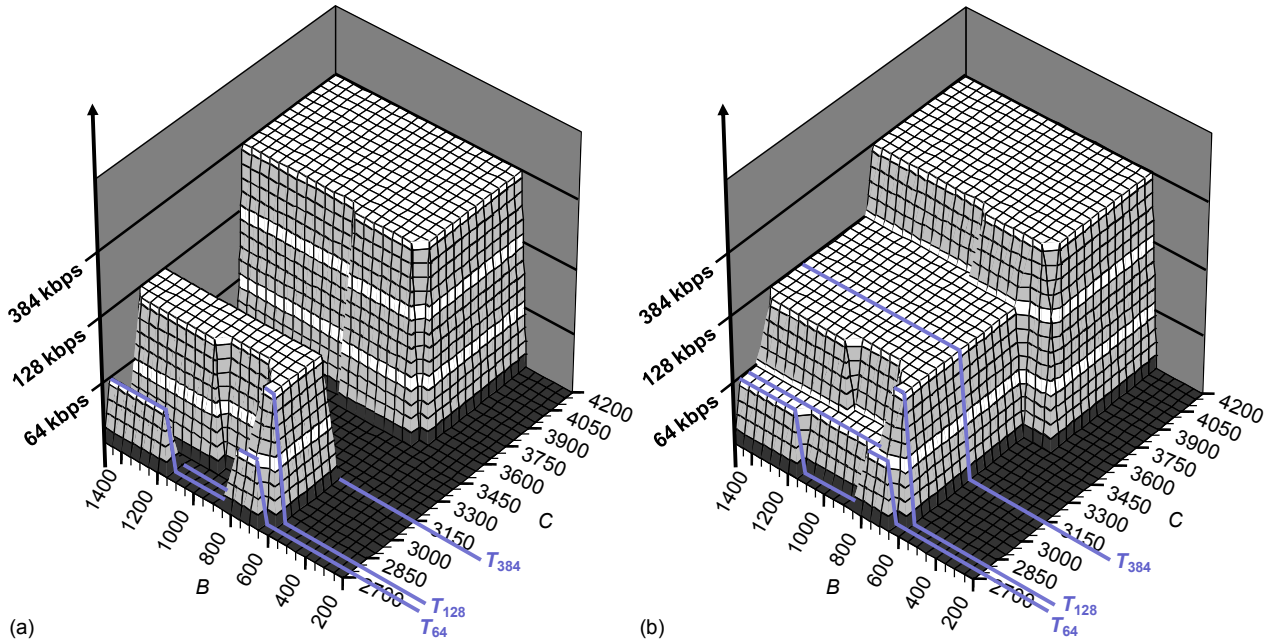


Fig. 7.12. Accumulated mapping information due to the learning process of Alg-1 (a) and Alg-2 (b) for SDR platform type XXIV.

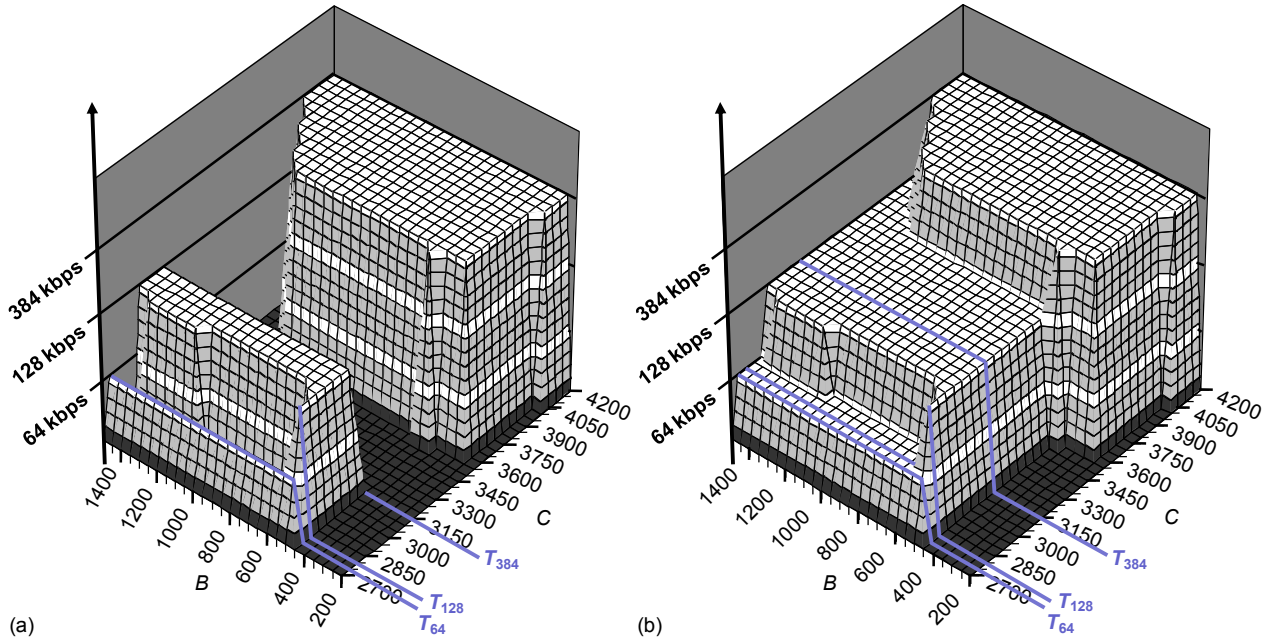


Fig. 7.13. Accumulated mapping information due to the learning process of Alg-1 (a) and Alg-2 (b) for SDR platform type XXV.

7.5 Summary

Cognitive radio is commonly considered as a *tool* for advanced RRM. Computing resource management has not been addressed so far in this context. We have argued for considering computing resource management as a cognitive task and correspondingly extended the scope of our SDR computing resource management framework.

There are many advantages of efficiently using the limited computing resource of SDR platforms, such as prolonging the battery lives of SDR-MTs. The awareness and learning capabilities of cognitive radio, moreover, facilitate a situation-dependent allocation and reallocation of computing resources. Knowledge may be locally obtained but should be globally distributed, accelerating the learning process through information sharing between cognitive radio entities or systems.

A cognitive computing resource management can increase the efficiency of individual computing equipment (selfish computing resource management) or maximize the utility of the overall computing system or environment (coordinated computing resource management). If computing resources are not shared, a selfish computing resource management may be appropriate. The need for resource sharing, on the other hand, calls for a coordinated computing resource management. Despite a few specific situations, such as emergency scenarios, a coordinated computing resource management should therefore be the ultimate goal. The following chapter deals with this from a more general perspective, proposing a joint management of the different resource types that facilitate SDR communications.

8

Joint Resource Management for Cognitive Radios

8.1 Introduction

The radio communications spectrum is commonly considered as a very scarce resource. Nevertheless, measurements have revealed that spectral resources are actually underused [142]. Technology advances and new RRM techniques facilitate a more efficient use of radio resources. More sophisticated digital signal processing techniques, though increasing the spectral efficiency, require more computing power. This has several implications on the computing resource management. Instead of merely enabling the implementation of advanced RRM methods, the cognitive computing resource management facilitates the efficient and situation-dependent use of the limited computing resources (Chapter 7). Trading radio against computing resources is then possible and may become the major achievement of cognitive radio.

We envisage cognitive radio systems that jointly manage all those resources that are required for SDR communications. We identify the *radio*, *computing*, and *application* resources as the three major resource types (Fig. 8.1). Radio, computing, and application resources basically refer to spectral resources, hardware capabilities, and the available software implementations of digital data processing algorithms (Section 8.2). Neither of these resource types can be directly quantified, although the underlying physical resources, such as processing powers and radio transmission bandwidth, can. Fig. 8.1 indicates orthogonality or independence between the three types of resources. Nevertheless, a reliable radio link requires a certain mix of radio, computing, and application resources.

Section 7.1 has indicated that cognitive radio research lacks computing resource management contributions. We therefore consider our cognitive computing resource management proposal of Chapter 7 as the basis for the computing and application resource management to be joined with the RRM. We intro-

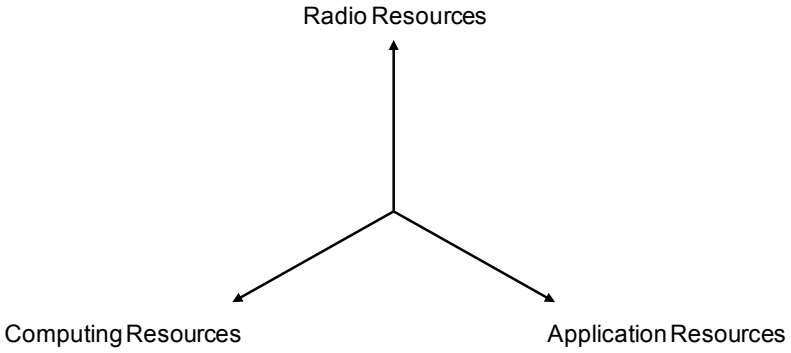


Fig. 8.1. Logical SDR resource space.

duce the cooperative and integrated resource management for jointly managing the radio, computing, and application resources (Section 8.3).

The computing resource management has so far implicitly managed the application resources. An explicit application resource management makes sense, though, because SDR facilitates a dynamic and modular SDR application design and update, the independent development of SDR applications and platforms, and their independent launch and deployment. It may then be more efficient to independently manage the complex computing and application environments (Section 8.4). This may further leverage the introduction of more sophisticated SDR platforms, more efficient SDR applications, and innovative user services. Despite this separation, the many correlations between all three resource types call for a joint management of the radio, computing, and application environments (Section 8.5).

8.2 Environments and Resources

We have already mentioned that different types of resource are necessary for modern radio communications. These include the available spectrum for radio communications, computing resources for processing signals before transmission and after reception, algorithms that define the signal processing functions, and applications that present the information to the wireless user. These resources are necessary for delivering user services and guaranteeing QoS. We can group them into four logical environments:

- the *radio environment* (RE),
- the *computing environment* (CE),
- the *radio application environment* (RAE), and
- the *user application environment* (UAE).

A mix of resources from these four resource pools can provide a wireless service at a certain QoS. From another viewpoint, *the service environment* (SE) represents a pool of *services* (and *QoSs*). Services may then be considered as resources as well. Fig. 8.2 illustrates the five logical environments, indicating some of their resources.

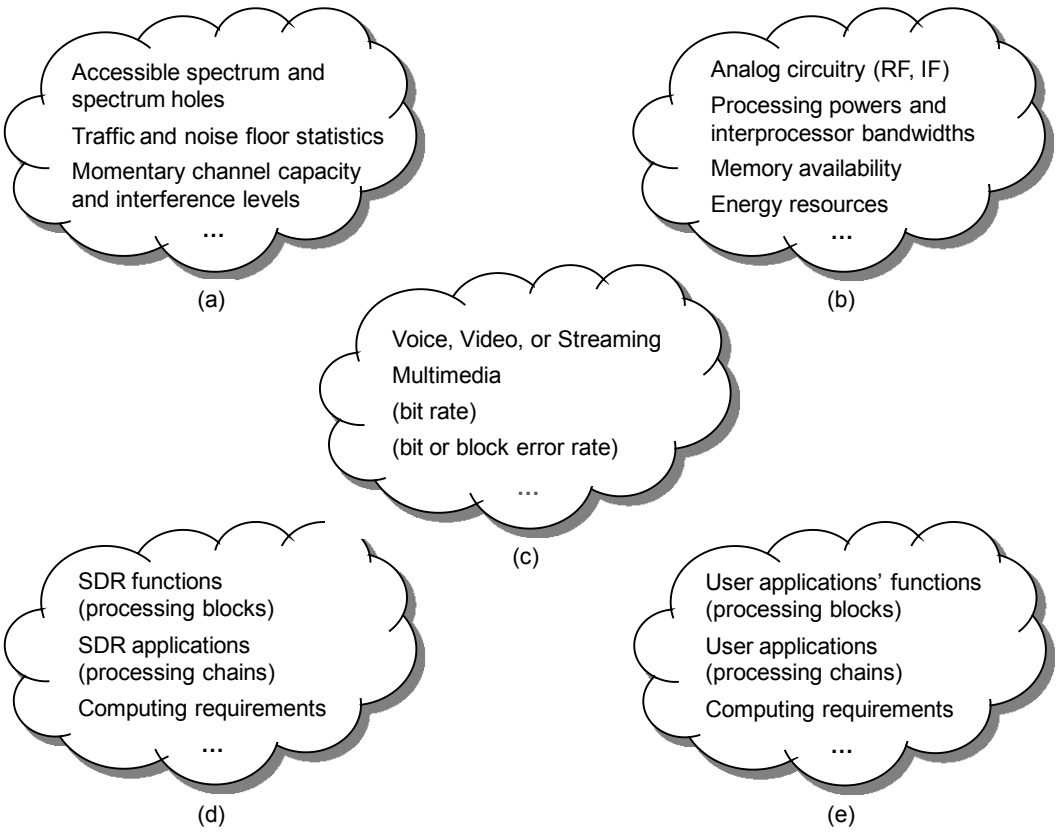


Fig. 8.2. Radio (a), computing (b), service (c), radio application (d), and user application (e) environments.

The CE here consists of the computing resources and hardware capabilities. It corresponds to the platform resources of the extended computing environment of Section 7.3.1. We use *radio application* synonymously to SDR application or waveform and group the RAE and UAE under the *application environment* (AE).

Radio applications contain the signal processing functions that facilitate the over-the-air delivery of services, whereas user applications specify the service presentations. Radio resources are required for over-the-air data transmissions and computing resource for the corresponding signal processing and the information presentation. Therefore, resources from the four principal environments─RE, CE, RAE, and UAE─are required for providing (communications) services over the wireless link. SDR and cognitive radio facilitate the flexible and intelligent usage of these resources. An efficient allocation of radio and computing resources through an intelligent selection of radio and user applications is then possible.

All five environments are highly dynamic because of the dynamic allocation and release of resources, steady upgrades of SDR platforms, constant improvements of signal processing algorithms, and evolving radio standards and user services, among others. Each environment, furthermore, consists of heterogeneous resources. This has several implications: Different transmission frequencies and bandwidths have different physical features which facilitate or impede a certain service delivery. Heterogeneous computing resources facilitate the execution of heterogeneous processing chains, but complicate the design of SDR frameworks and abstraction layers (Chapter 2). Service differentiations and personalized service provisioning, finally, require personalized agreements between service providers and end users and the management of many different user profiles.

Apart from the above issues, the limited amounts of resources constrain the capacity of the wireless access: Quality requirements and interference mainly limit the availability of radio resources, whereas the computing capabilities restrain the execution of applications and, thus, the service and QoS delivery. The wireless subscriber becomes aware of these constraints through QoS degradations, service interruptions, or the lack of service versatility, among others. One major task of the joint resource management is then making these resource limitations transparent to the wireless user.

8.3 Cooperative versus Integrated Resource Management

A joint resource management can be implemented in many ways. We suggest either a distributed or centralized resource management and introduce the cooperative and integrated resource management concepts. The following definitions point out the main differences.

Definition 5 Cooperative resource management: *The separate resource management entities interchange their individual objectives and decisions, which are then adjusted for the system's overall benefit.*

Definition 6 Integrated resource management (IRM): *One resource management entity processes all environmental information for deriving the appropriate actions that maximize the system's overall benefit.*

These definitions indicate the existence of distributed resource management entities that cooperate with one another (Fig. 8.3a, Table 8.1) or a central entity, which implicitly processes the all environmental information (Fig. 8.3b, Table 8.1). A cooperative resource management may be easier to develop and implement, whereas an integrated resource management may provide better results. The selection of one or another approach is a function of the environmental conditions, management policies, and so forth. Rather than discussing these issues, we introduce general concepts for the joint resource management, be it cooperative or integrated.

Although we may logically group and independently model the different resource types, the corresponding environments overlap in practice. Resources from different types are, moreover, correlated: A radio or user application specifies the minimum amount of computing resources that it requires for execution. The computing requirements are, generally, a function of the service and QoS demands (AE ↔ SE) and the given radio conditions (AE ↔ RE). The selection of the appropriate applications is, finally, a function of the available computing resources (AE ↔ CE).

Table 8.1 Resource management abbreviations.

CRM$_{(D)}$	Computing resource management (for platform *D*)
JARM	Joint application resource management
JCRM	Joint computing resource management
JRARM	Joint radio application resource management
JRRM	Joint radio resource management
JUARM	Joint user application resource management
RARM$_{(d)}$	Radio application resource management (for radio application *d*)
RRM$_{(Z)}$	Radio resource management (for RAT *Z*)
UARM$_{(d)}$	User application resource management (for user application *d*)

The joint resource management can account for these interrelations and take advantage of them. It, particularly, facilitates selecting the radio and user applications for each SDR platform as a function of the radio, computing, application, and service environments. We, therefore, argue for a cognitive radio system that implements either a cooperative or an integrated resource management. For resources to be managed, their status needs to be known at any time. This requires a suitable resource modeling and monitoring, which is presented in continuation.

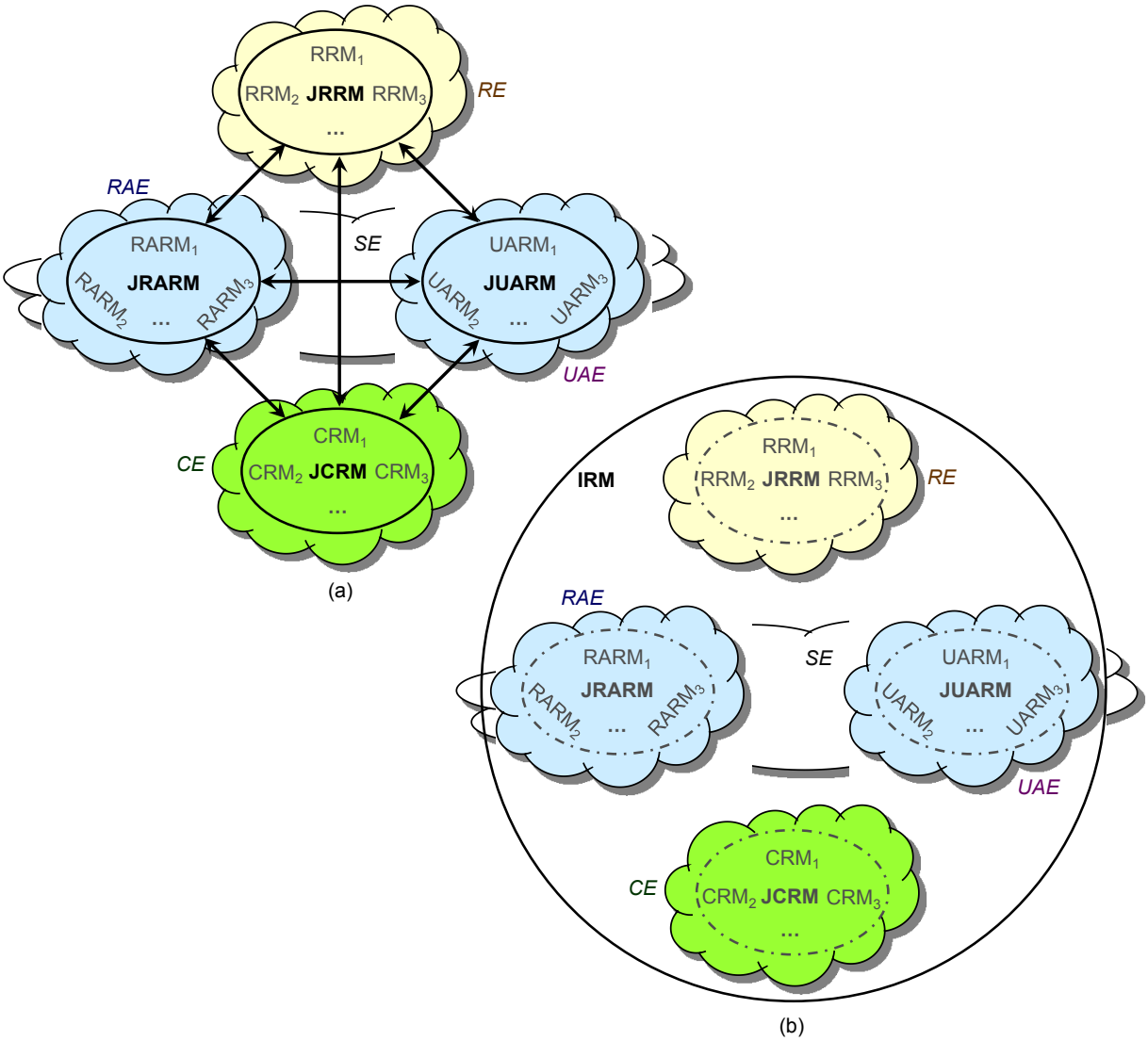


Fig. 8.3. Cooperative (a) and integrated (b) resource management concepts.

8.4 Cognitive Cycles

This section presents three cognitive cycles and establishes a simple modeling of the radio, computing, and application environments. Sections 8.4.1 to 8.4.3 assume the cooperative resource management context. Section 8.4.4 discusses the joint resource management implications.

8.4.1 Radio Cycle

Radio resource management is currently very important and the major research issue in the context of cognitive radio [116]. Apart from the rising number of wireless subscribers and their constantly increasing bandwidth demands, the observations that many radio frequency bands are underused [142] underlines the importance and the potentials of the JRRM in heterogeneous radio environments [132]. Recent cognitive radio research therefore tries to achieve a more efficient spectral usage and, particularly, evaluates different dynamic spectrum access techniques (Section 1.4.1B)).

Cognitive radio facilitates a situation dependent management of spectral or radio resources. The *radio cycle* (Fig. 8.4) therefore continuously monitors the radio environment and decides upon the most appropriate radio resource allocation at any time. The radio scene analysis module, particularly, updates information about owned and leased radio frequencies and their occupations, whereas the JRRM entity uses this information for evaluating and triggering radio resource allocations or reallocations. This way the radio resources can be dynamically managed for increasing the overall system capacity, radio operator revenues, or the like.

The basis for an efficient information processing is a suitable modeling. In this case, the modeling needs to capture the channel characteristics. Xing et al. [130] illustrate how a frequency band can be divided into channels. It shows two frequency grids, one for a narrowband and one for a wideband radio system. Since the grids overlap, the frequency grid of the narrowband system specifies the channel granularity. We adopt this approach, although we do not require that the entire spectrum be divided in channels of equal bandwidths, but, rather, that a channel's bandwidth be a function of the radio systems that may operate in the corresponding band. The channel granularity may, furthermore, be dynamically adjustable. A channel may then be divided or adjacent channels merged, facilitating an efficient management: the smaller the channels the more flexible the spectrum allocation, but the more complex the JRRM.

We assume that spectrum leasing is possible, whereas spectrum bands may or may not be preassigned to certain RATs. A channel is described by several parameters, such as *owner*, *leaser*, and *occupation*. If a channel is licensed, its owner is the license holder; otherwise, the owner is *public*. The occupation metric could, for instance, relate the actual to the tolerable interference level of a channel. It may take discrete or continuous values. In the former case we may specify a channel's occupation as *free*, *low*, *medium*, or *high* and, in the latter, through the occupation percentage. The radio environment can then be modeled as

$$(\boldsymbol{R}_{\mathrm{R}})^{ch} = (\text{<start frequency [MHz]>, <end frequency [MHz]>, <owner>, <leaser>, <occupation [\%]>}), \quad (8.1)$$

where $ch \in \mathbb{Z}$ (integers) specifies the channel number.

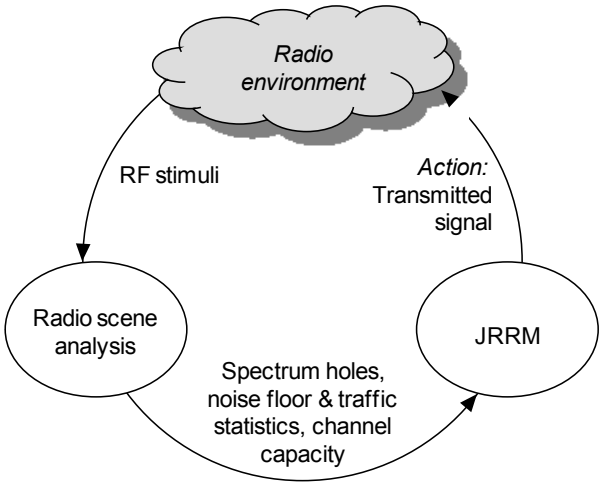


Fig. 8.4. The radio cycle.

The channel numbering could be arbitrary. A logical numbering, where channel n is adjacent to channels n–1 and n+1, would, however, simplify finding adjacent channels. We assume that the system knows about the RAT suitability for a given channel and the number of (adjacent) channels required for transmission. These assumptions will be relaxed in future versions of (8.1). Based on the momentary radio resource states, primarily described by the fifth element of $(\boldsymbol{R}_{\mathrm{R}})^{ch}$, the JRRM entity would then be able to identify over and underused channels. This information could serve for distributing radio resources and increasing the spectral efficiency.

The above model is simple and flexible. We can extend it vertically and horizontally for tracking additional channels and channel information. Channels can also be easily be split or merged. This is in line with the modeling concept of Chapter 3. $(\boldsymbol{R}_{\mathrm{R}})^{ch}$ could, particularly, be considered as an instance of a general radio resource modeling template that facilitates accounting for any relevant radio resource information.

8.4.2 Computing Cycle

SDR characterizes reconfigurable radio equipment that consists of general-purpose and software-reconfigurable processors, such as DSPs and FPGAs. Any processing entity has a limited amount of computing resources, including processing powers, interprocessor bandwidths, and energy capacities. Computing resource management is therefore necessary and facilitates dynamically switching the deployed radio communications standard. The computing environment is therefore characterized by the available and occupied computing resources, the RF circuitry, and so forth.

Cognitive radio automates the reconfiguration process of SDR platforms and is, moreover, capable of dynamically managing the computing environment. We therefore suggest the *computing cycle* of Fig. 8.5, which monitors the states of the reconfigurable computing resources and hardware capabilities (c*omputing scene analysis*) for the joint computing resource management (JCRM). More precisely, the computing scene analysis entity continuously observes the computing environment and provides any changes to the JCRM module. The JCRM module processes this information before deciding which SDR platforms to reconfigure, if any. This computing cycle differs from the cognitive computing cycle of Fig. 7.3 in that it exclusively addresses the hardware resources.

Computing resource management is part of the radio infrastructure, the interface between radio applications and hardware platforms (Section 7.3.2). Although conceptually independent of the particular hardware platform, we consider it efficient to provide computing resource management facilities locally, that is, on SDR platform basis.

The computing scene analysis entity provides the model of the computing environment, which is processed by the JCRM module. It should contain all relevant hardware information and be easily manageable. A simple model would particularly ease the dynamic and efficient tracking of hardware capabilities and computing resources, instantly informing about any state changes. Section 7.3.4 has discussed the suitability of our SDR computing system modeling of Chapter 3. Since we distinguish between the com-

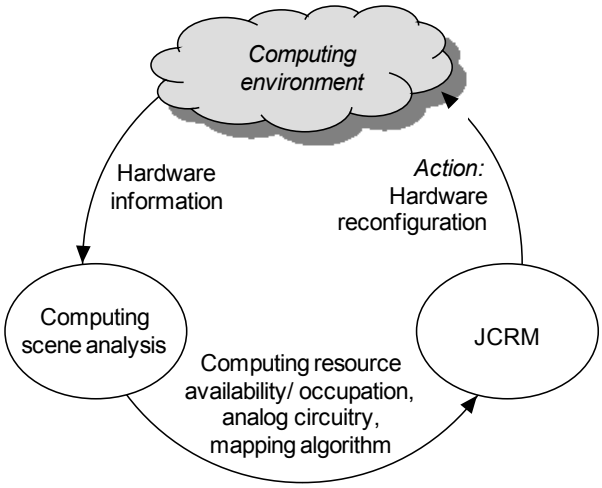


Fig. 8.5. The computing cycle.

puting and application resource management here, only the platform modeling of Sections 3.3.1 and 7.3.4A) applies.

8.4.3 Application Cycle

Radio and user applications basically differ in their utility. Radio applications define the radio functionalities of SDR platforms and, thus, the data transmission and reception modes. They consist of signal processing functions (SDR functions) that specify the modulation scheme and the error correction algorithms, among others. User applications, on the other hand, facilitate the use of services, providing (personalized) service or data presentations. A user application thus captures and presents information before and after being processed by a radio application, respectively. User and radio applications together provide the communications services.

Both application types will be modularly built out of precedence-constrained processing blocks that receive, process, and propagate data. This facilitates the distributed execution and modifications of (individual) processing blocks.

We expect that the radio application environment will contain a number of SDR applications, supporting many different (multimedia) services and QoS levels. The inherent modularity of SDRs augments the number of possible SDR applications, resulting in a highly dynamic radio application environment. An SDR function can often be implemented in several ways. Many functions, moreover, embody dynamic algorithms with highly variable computing requirements. The processing requirement of the turbo decoder, for instance, is a function of the number of iterations, which is dynamically adjustable. The evolution of RAT components, such as new coding techniques, increases the diversity, but, also, the complexity of the radio application environment.

The user application environment will correspondingly consist of many user applications. An object-oriented design permits offering many different and personalized service presentations, enriching the user application environment and increasing the service values.

Software-defined signal processing chains for many RATs may be available on special application servers. It is not envisaged that more than one SDR application, the active one, will be locally available on an SDR-MT, because of the terminal's memory constraints and the dynamic radio application environment. Therefore, before the reconfiguration of an SDR-MT, the new RAT-software is downloaded from the network using, for example, a radio link [125].

User applications will also be available for download from servers. Many radio-user application pairs may be valid. The radio application is principally chosen as a function of the desired service and QoS. The user application is chosen as a function of the service type and the user preferences. The preferred user application may have implications on the selection of the radio application and vice versa. Therefore, a valid combination between a radio and user application should be found before initiating any software download.

The above discussion suggests that both application environments be continuously scanned for availa-

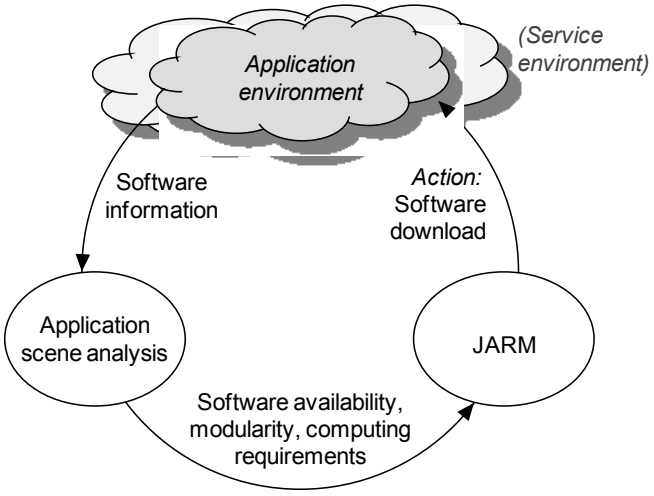


Fig. 8.6. The application cycle.

ble applications and their modules. The *application scene analysis* entity collects this information, processes it, and provides it to the *joint application resource management* (JARM) entity. The JARM entity then specifies the necessary software downloads. This corresponds to the application cycle of Fig. 8.6, which is representative for the radio application and user application cycles.

The cognitive radio system may execute two application cycles, one for the radio and one for the user applications. Although the radio and user applications have different functionalities and requirements, the same modeling, presented in Section 3.3.2 and extended in Section 7.3.4B), may serve for both. The previously introduced models capture some application characteristics and computing requirements. The application cycle, though, also needs to process information about the suitability of an application for a given service. This may be modeled as another instance of $(\boldsymbol{R}_\mathrm{A}^{t''})^{\boldsymbol{d}}$ (3.8):

$$(\boldsymbol{R}_\mathrm{A}^{9})^{\boldsymbol{d}} = \boldsymbol{qos}^{\boldsymbol{d}} = ((qos_1)^d, (qos_2)^d, \ldots, (qos_K)^d). \tag{8.2}$$

$(\boldsymbol{R}_\mathrm{A}^{9})^{\boldsymbol{d}}$ indicates the ninth instance of the general application template $(\boldsymbol{R}_\mathrm{A}^{t''})^{\boldsymbol{d}}$ (Section 3.3.2A)). Element $(qos_k)^d \in \mathbb{R}^+$ specifies the relative QoS that application d can provide for service k as a function of the achievable bit rate, BER, or the like.

For each service (type), we suggest defining a reference QoS value, for example, $(qos_\mathrm{video})^{\text{128kbps UMTS}} = 1$. Since we do not scale $(qos_k)^d$, we can freely choose this reference, simplifying the definition and update of $\boldsymbol{qos}^{\boldsymbol{d}}$ due to future QoS enhancements, for example. If $(qos_k)^d = 0$, however, service k is not supported by application d. K stands for the number of user or radio applications. It will grow with each new application that is introduced, requiring an according update of the *QoS suitability model* (8.2).

8.4.4 Joint Resource Management Implications

We have introduced three cognitive cycles that monitor the radio, computing, and application environments and manage the corresponding resources. The cycles create and update the environmental models, which capture the states of all relevant resources and requirements. The management entities process this information before specifying the most appropriate resource allocations or deallocations. The distributed entities therefore cooperate with one another, coordinating their decisions rather than making individual and selfish decisions. In other words, they cooperatively manage the different resource types.

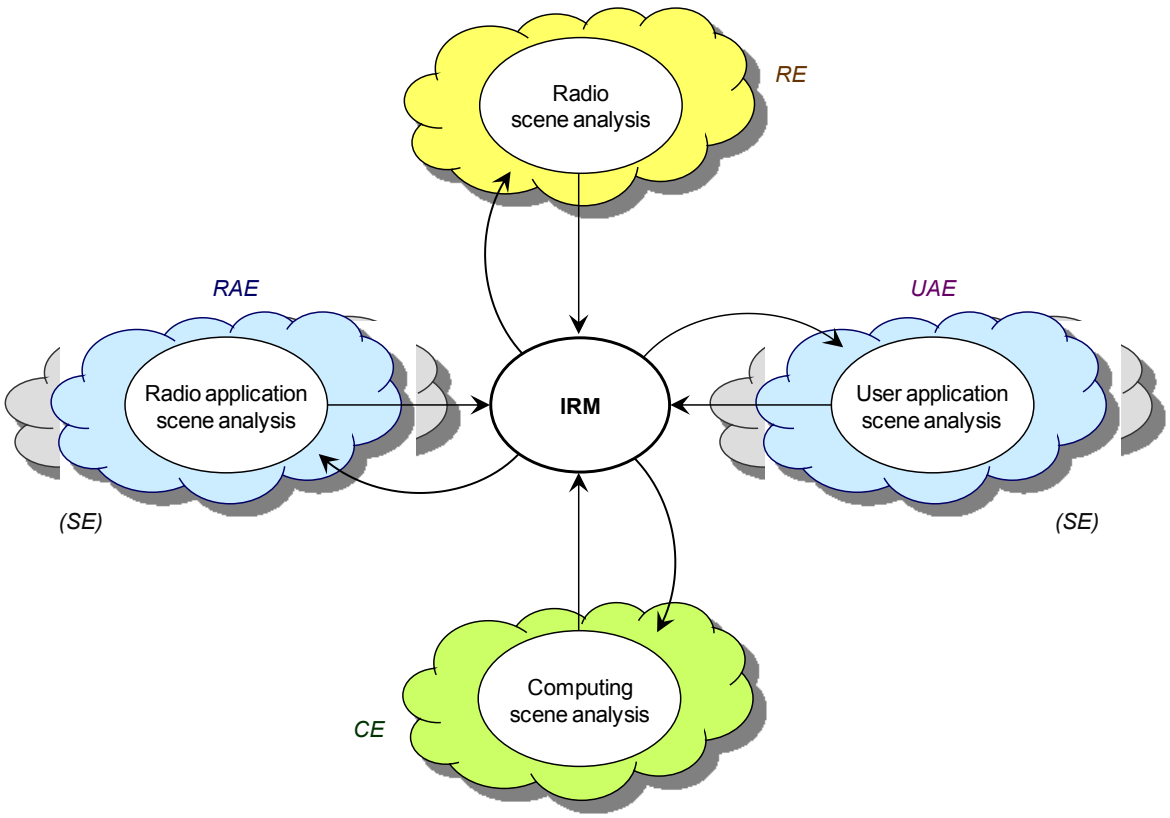


Fig. 8.7. The cognitive cycles within the integrated resource management context.

Fig. 8.7 illustrates the four cognitive cycles within the integrated resource management context. Each scene analysis entity collects and processes the information about the corresponding environment, before providing it to the IRM entity. The IRM entity synchronizes and jointly processes the received environmental information. Its resource management decisions will, generally, affect several environments and lead to new resource management stimuli.

8.5 Proof of Concept

This section aims at demonstrating the suitability of the above concepts. It discusses several interrelations between the different environments and resource types (Section 8.5.1) before providing the first simulation results that indicate the utility of the joint resource management concept (Section 8.5.2).

8.5.1 Discussion

Cognitive radio facilitates maximizing the benefits or revenues of all parties involved in modern wireless communications [112]. The wireless subscriber is one such party and his satisfaction may become other parties' revenues. Radio operators are therefore eager for maximizing the number of users they can simultaneously serve. Each user that is not admitted due to network capacity limits is lost revenue because the user session cannot be initiated and charged and because the user may switch to another operator in the long term. Hence, radio resource management is essential in cognitive radio [138].

Service providers also compete for users. They need to introduce innovative user services that attract as many users as possible as well as offer personalized services. The evolution of SDR technology will lead to more sophisticated user applications while increasing the service diversity. The user application resource management facilitates managing this increasing amount of user applications.

Once the first radio applications become available, there will be no limit in designing new and optimized RAT implementations. Many parties may participate in the design of radio applications. The radio application resource management will handle the growing amount of radio applications and assists in the selection of the radio software downloads.

Table 8.2 Resources, optimization goals, and resource correlations.

	Resources	Optimization goals	Inter-environmental resource correlations
Radio environment	Spectrum Transmission power	Maximize the overall system capacity (minimize or distribute interference).	More wireless users = more interference (RE) => better receivers required (RAE) and more computing power needed (CE)
Computing environment	Analog circuitry DSP modules: topology, processing powers, bandwidths, memory, etc. Energy resources Application mapping	Efficient use of computing resources. That is, 1. distribute the computing loads to meet the RAT-specific computing requirements (real-time processing and latency), 2. minimize the power consumption, and so forth.	1. Some applications may not be feasibly mappable to the available computing resources (RE, AE, CE). 2. Less complex applications are desired to minimize the power consumption, among others (AE, CE)
Application environment	Applications Applications' modules (functions) Computing requirements	Select the radio and user applications that best meet the user preferences and QoS requests.	Although many applications may be suitable (AE), the radio conditions (RE) and computing capabilities (CE) constrain their selection.

Computing resource management is essential for supporting service delivery whenever possible and as long as desired. Each computing operation, however, consumes power and costs money. Hence, computing efficient mappings of user and radio applications are essential for direct and indirect revenues. Computing operators or radio operators dealing with computing resource management issues will be responsible for that.

Table 8.2 summarizes the principal resources of the radio, computing, and application environments, the optimization goals, and some inter-environmental resource correlations. It reveals that, although we can formulate an optimization objective for each environment individually, there are many inter-environmental resource dependencies and compatibility issues. Many parameters influence the selection of radio, application, and computing resources. Therefore, instead of managing each environment separately, we argue for the joint resource management across environmental boundaries.

We conclude that future wireless communications will offer a very wide variety of user services at different QoS levels. More than one RAT will, moreover, be capable of meeting most service and QoS requests. Many wireless users are not interested in the specific air interface that provides the service, as long as it is affordable and of the desired quality. RATs can then be flexibly chosen in this world of service-driven wireless communications. A cognitive radio system with full resource control can, moreover, distribute the resource loads and trade off one resource type against another as a function of the radio, computing, application, and service environments. A joint resource management potentiates such resource tradeoffs.

8.5.2 Simulations

The context of this section could be the following scenario: User *A* has established a voice session with user *B* using the GPRS RAT. During the course of the conversation a switch to a video conference is desired without interrupting the session. The users thus notify the network of the desired service upgrade.

The system evaluates the applicability of the available SDR applications that support the desired video service. It realizes that user *A* could be given the services at an adequate QoS using either UMTS or WLAN. The WLAN implementation requires less computing resources than the available UMTS implementations. The WLAN implementation is therefore chosen. User *B* is not within the reach of a WLAN hot spot. Hence, the system decides to use the WCDMA air interface. Since being mobile terminals with a small display, the 128 kbps data rate would be enough.

The 128 kbps UMTS transceiver implementation is downloaded from a radio application server to the mobile terminal of user *B*, whereas user *A* receives a WLAN processing chain. Each radio application is then internally mapped to the available computing resources of the corresponding SDR platform in real-time and without interrupting the voice session. Once both terminals operate in the new modes, they can smoothly switch from voice conversation to videoconference.

Motivated by the above example, we simulate a basic heterogeneous radio scenario and apply a simple IRM algorithm for proving the concepts of this chapter. The simulations analyze the effect of integrating the three cognitive cycles of Section 8.4 on the intelligent reconfiguration of SDR-MTs.

A) Radio Scenario

We simulate a heterogeneous radio scenario, where the cognitive radio system controls an area with GPRS and UMTS coverage (Fig. 8.8). SDR-MTs enter, remain, and exit the system. Each user initiates or runs a session in the GPRS mode and observes up to three vertical handovers (from GPRS to UMTS or vice versa), horizontal handovers (between the two Node Bs), or QoS up or downgrades (through data rate adjustments) as a function of the service request and the terminal position. The mode probabilities are 0.4 for GPRS, 0.3 for 64 kbps UMTS, 0.2 for 128 kbps UMTS, and 0.1 for 384 kbps UMTS. A vertical handover or QoS up or downgrade requires a terminal reconfiguration. If this fails, the user session is lost. The scenario assumes that the UMTS and GPRS radio resources are available.

Through random variables and the above mode probabilities we simulate different user positions (Fig. 8.8) and QoS requirements. More precisely, a random variable takes one out of four values based on the given mode probabilities. It specifies the target mode of an SDR-MT. Three successive target modes per terminal are assumed. A target mode that does not coincide with the terminal's running mode triggers a horizontal handover, a vertical handover, or a QoS up or downgrade. A horizontal handover is assumed to be always possible, whereas a vertical handover or QoS up or downgrade, requiring reconfiguration, is

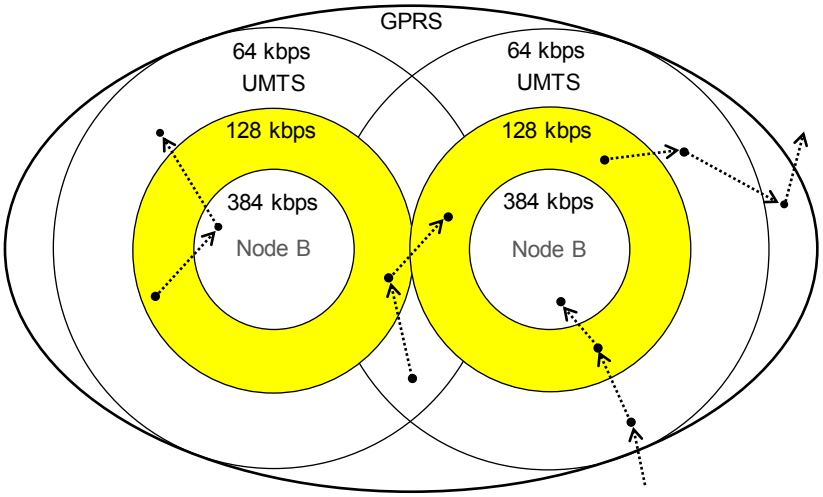


Fig. 8.8. Radio scenario – the coverage area of a cognitive radio system.

evaluated by resource management entity (Section 8.5.2C). This way each terminal is reconfigured zero, one, two, or three times.

B) Computing and Application Characteristics

We consider 5000 SDR-MTs, which indicate the simulation time span. These terminals have the same communication architecture but offer different amounts of computing resources (Fig. 8.9). $(C_1)^D$, $(C_2)^D$, and $(C_3)^D$ are the processing capacities of processors $(P_1)^D$, $(P_2)^D$, and $(P_3)^D$ and are, for simulation purpose, independent random variables that are uniformly distributed in {2700, 2800, …, 3800} MOPS. B_{bus} = 3000 Mbps is the bus bandwidth that facilitates bidirectional data transfers between any two processors. The 4-tupel $\{(C_1)^D, (C_2)^D, (C_3)^D, B_{bus}\}$ describes platform D's computing capacities after the (projected) demapping of the SDR functions of the currently running mode.

We assume that each terminal contains the analog RF circuitry that is necessary for accessing the different air interfaces and suppose full network support. The P-HAL-OE (Section 2.2.3) and the t_1-mapping algorithm with cost function (4.9) (Sections 4.2.1 and 4.3.2) facilitate the reconfiguration of these SDR platforms.

The radio applications are the three software-defined UMTS receivers of Section 7.4.2C), supporting the raw data rates of 64, 128, and 384 kbps. Their total processing requirements $(c_T)^{64kbps}$, $(c_T)^{128kbps}$, and $(c_T)^{384kbps}$ are approximately 8130, 8300, and 9400 MOPS (Section 7.4.2C)). We assume that the processing and bandwidth resources $(C_1)^D$, $(C_2)^D$, $(C_3)^D$, and B_{bus} are dedicated for executing these SDR applications and that additional computing resources facilitate the mapping of the UMTS or GPRS uplink transmitter, the remaining receiver functions, and the user application.

C) Resource Management Algorithms

We apply two resource management algorithms, a simple IRM algorithm (IRMA) and a baseline algorithm (BSLA). As opposed to the IRMA, the BSLA considers only the radio cycle and is unaware of the computing resource states and the application resource requirements.

Baseline Algorithm

The BSLA, implemented within the JRRM entity of the radio cycle, assumes the availability of radio applications and full computing resource support. It selects the radio application as a function of the radio scene analysis. Despite ignoring the computing and application cycles, the BSLA could still implement a

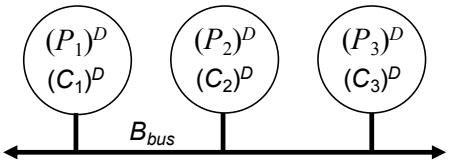


Fig. 8.9. SDR platform architecture XXVI.

cognitive learning algorithm that processes the available information about the radio environment. These algorithms are not considered here; we, rather, assume that enough radio resources are available.

Integrate Resource Management Algorithm

The IRMA reconfigures an SDR-MT as a function of the platform's computing capacity (CE), the available SDR applications and their computing requirements (RAE), and the radio resource availability at the user's current position (RE). The IRME initially defines the following reconfiguration rules as a function of a platform's processing capacity $(C_T)^D$ and the SDR applications' processing requirements $(c_T)^{64kbps}$, $(c_T)^{128kbps}$, and $(c_T)^{384kbps}$:

- 384 kbps UMTS if $(C_T)^D \geq (c_T)^{384kbps}$,
- 128 kbps UMTS if $(c_T)^{384kbps} > (C_T)^D \geq (c_T)^{128kbps}$
- 64 kbps UMTS if $(c_T)^{128kbps} > (C_T)^D \geq (c_T)^{64kbps}$, and
- GRPS if $(c_T)^{64kbps} > (C_T)^D$.

The IMRA chooses the best possible solution as a function of the above rules, the radio scene analysis, and the accumulated mapping information. The best possible solution is selecting the SDR application that supports or nearly supports the desired QoS. We assume that the users are willing to accept a possible QoS degradation whenever the desired radio mode is not computing feasible.

A mapping of an SDR application to an SDR platform is either feasible or infeasible. If a reconfiguration is initiated but results in an infeasible mapping, the session is lost. The JCRM or IRM module will then consider the SDR application with the next lower processing requirement for the following platform with identical computing resource characteristics. Hence, the cognitive radio system continuously refines its database of feasible and infeasible platform-application pairs. This algorithm corresponds to Alg-2 of Section 7.4.2D).

Due to Section 3.3.1B), the internal SDR platform modeling assumes $(C_1)^D \geq (C_2)^D \geq (C_3)^D$. The symmetry of the platform architecture of Fig. 8.9 then requires testing only about a sixth part of the different SDR-MTs for compatibility with the SDR applications. This accelerates the cognitive learning process and minimizes the memory demand for the knowledge acquisition.

D) Simulation Results

Fig. 8.10a shows the evolution of the number of lost sessions resulting from infeasible reconfigurations. We observe that the curve corresponding to the BSLA continuously increases, whereas the learning capability of the IRMA leads to saturation, which begins around the 2000th SDR-MT. This specifies the cognitive radio system's time for adapting to its environment. More sophisticated IRMAs or more initial information could achieve faster adaptations. Fast adaptations are necessary for keeping pace with the steadily increasing dynamism of the radio, computing, and application environments.

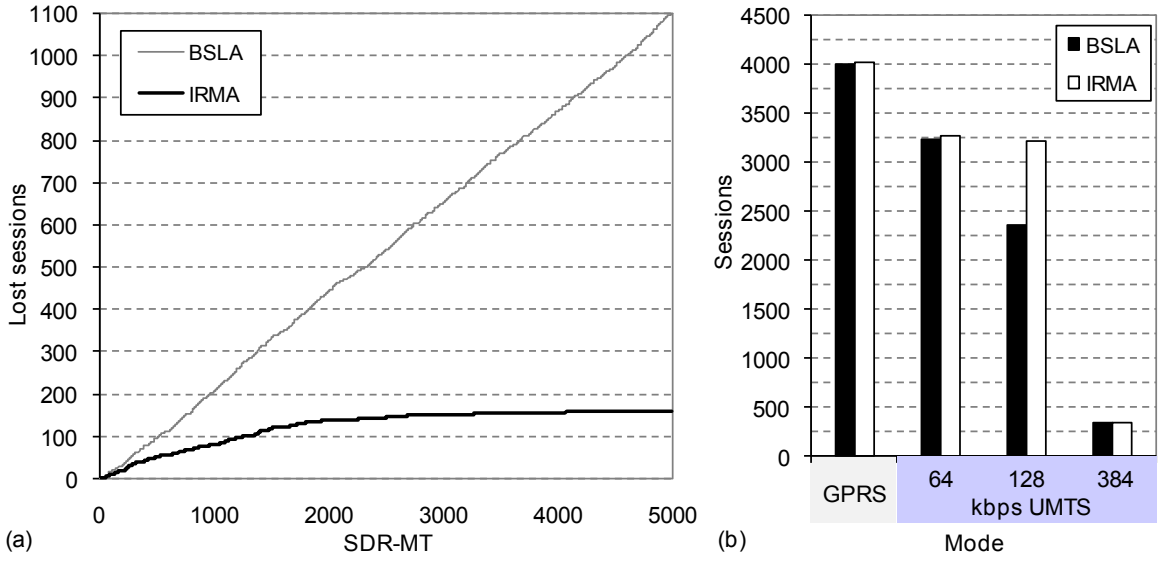


Fig. 8.10. Evolution of the number of lost sessions (a) and total number of feasible reconfigurations (b).

The total number of terminal reconfigurations is 11 006; 9907 and 10 847 are feasible with the BSLA and IRMA. Fig. 8.10b shows the total number of sessions per mode that followed a feasible reconfiguration. We observe that the number of feasible reconfigurations due to the IRMA is, for any mode, equal or higher than that of the BSLA. This means that the IRMA sacrifices QoS only for the sake of connectivity. In other words, whenever the new mode that the radio scene analysis suggests for a given SDR-MT is computing feasible, the IRMA reconfigures the terminal to this mode.

The highest difference in the number of feasible reconfigurations is observed for the 128 kbps UMTS mode (Fig. 8.10b). This is so because only 339 of the approximately 1500 reconfiguration requests to the 384 kbps UMTS mode are feasible. Since the IRMA loses a total of 159 sessions, at most 159 reconfiguration intents to the 384 kbps UMTS mode are infeasible. Fig. 8.10a and b indicate that the IRMA accomplishes to feasibly reconfigure most of the SDT-MT that cannot operate in the 384 UMTS mode to the 128 kbps mode, which requires considerably less computing resources due to Sections 8.5.2B) and 7.4.2C).

The gap between the two resource management algorithms is relatively low for the 64 kbps UMTS mode, because of the similar processing demands of the 128 and 64 kbps UMTS modes. The negligible difference for the GRPS mode implies that most simulated SDR-MTs provide enough computing resources for implementing the 64 kbps UMTS receiver. This, moreover, indicates that the majority of the reconfigurations to the 128 kbps UMTS mode are feasible and that the lost sessions primarily stem from the infeasible mappings of the 384 kbps UMTS receiver processing chain.

8.6 Summary

Motivated by the conclusions of Chapter 7, this chapter has discussed a new management approach for cognitive radios. The *joint resource management* concept accounts for all types of resources that are necessary for SDR communications, including the radio, computing, and application resources. We have discussed some cooperative and integrated resource management peculiarities and introduced three cognitive cycles and the respective resource modeling.

Resources may be considered in common resource pools: radio resources in a radio resource pool, computing resources in a computing resource pool, and application resources in an application resource pool. These pools should be either cooperatively managed or considered as a single resource pool for management purposes. This chapter pointed out several advantages of a joint resource management, be it cooperative or integrated.

We conclude that the higher the resource diversity, the more important the resource management. The possibility of resource sharing or trading further increases the management flexibility, but also its complexity. Only a joint resource management can account for the correlations between resources of different types and facilitate flexible allocations. A momentary increased consumption of computing resources while decreasing the need for radio resources and vice versa may then be possible.

9

Conclusions

9.1 Contribution

The constant evolution of RATs, radio services, and computing devices, among others, make the wireless environment highly dynamic and unpredictable. SDR introduces flexibility to wireless communications. It requires a flexible computing resource management framework that can deal with real-time computing constraints and changing QoS demands of wireless systems. The principal contribution of this dissertation is an SDR computing resource management framework that facilitates dynamic reconfigurations of SDR platforms (Fig. 9.1).

Our framework consists of two parts: an *SDR computing system modeling* that facilitates the *SDR computing resource management*. The system modeling accounts for the distributed and limited computing resources of SDR platforms and the real-time computing requirements of SDR applications. Its modular design eases the monitoring of the relevant platform and application characteristics, including computing resources and requirements. Our computing resource management contribution distinguishes between the mapping algorithm and the 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 the computing resources of SDR platforms for satisfying the computing requirements of SDR applications.

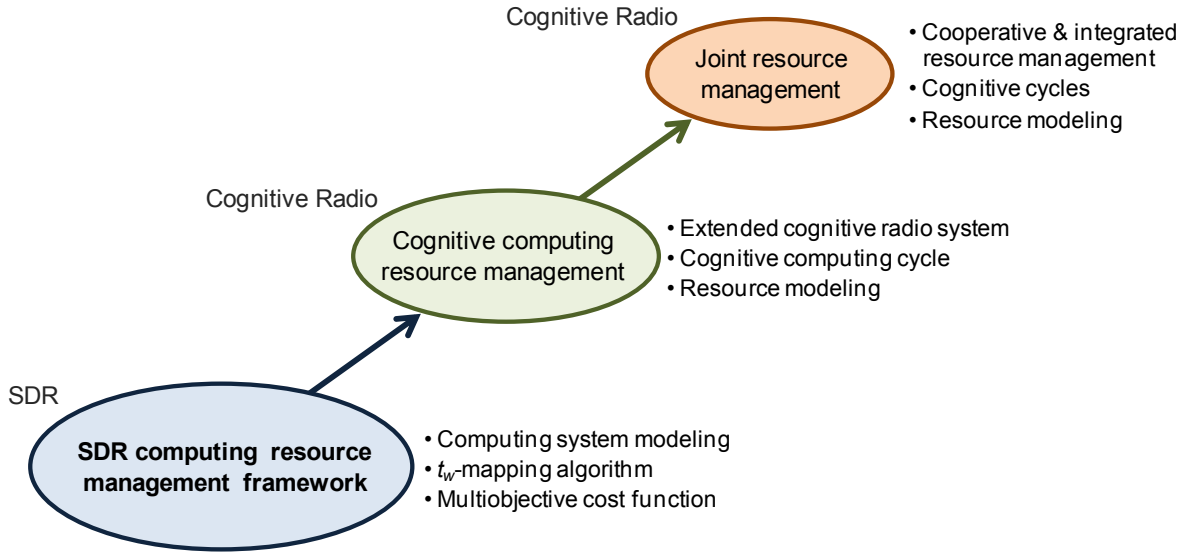


Fig. 9.1. The evolution of this dissertation and its principal contributions.

We have simulated two SDR scenarios for demonstrating the appropriateness of the entire framework. We have observed that the t_w-mapping, as opposed to a baseline algorithm, achieves feasible and even optimal results for window sizes as small as 1, 2, or 3. Further simulations have revealed some interrelations between the hardware architecture and the application mapping as well as indicated the importance of the mapping order, the cost function parameter, and the window size. We have shown that the t_w-mapping is fast enough for real implementations, predictable in terms of computing complexity and mapping result, and suitable for different waveforms and platforms. Our framework is, hence, flexibly adjustable for solving different reconfiguration scenarios in heterogeneous and nonstationary radio and computing environments.

After having demonstrated the suitability of our computing resource management proposal for SDRs, we have analyzed its cognitive capabilities. We have introduced the cognitive computing cycle for monitoring and managing the computing environment of future cognitive radio systems. Our SDR computing system modeling and computing resource management contributions facilitate continuously monitoring and managing the hardware capabilities and the software processing requirements.

Cognitive radio facilitates a flexible management of all types of resources that are necessary for SDR communications. These include the radio as well as the computing and application resources. We have, therefore, presented the joint resource management concept and three cognitive cycles that cooperatively manage the radio, computing, and application resources. Fig. 9.1 summarizes the evolution of the dissertation and its main contributions.

9.2 Future Work

In the SDR computing context, it remains to analyze other cost functions and further examine how to adjust the computing resource management parameters to the particular reconfiguration scenario. Dynamic adjustments of these parameters during the mapping process may also be conceivable and will be explored.

Furthermore, we need to simulate additional SDR scenarios and, specifically, analyze the management of computing resources at multi-user base stations. A computing efficient management of large arrays of processors may require a distributed t_w-mapping implementation or clustering the processor array. We will examine both techniques and their implications on real-time computing. Other heterogeneous computing methods will also be explored in this context.

We will study the impact of hardware abstractions on the system performance and fully implement our computing resource management framework within the P-HAL-OE. We can then test P-HAL-OE and its computing resource management capabilities as a function of the hardware platform and software implementation. This requires multidisciplinary research efforts; we have, therefore, recently created a platform for collaborative SDR and cognitive radio research under the FlexNets (flexible wireless systems and networks) initiative [143].

FlexNets is an open source initiative under the GNU license. Its objective is integrating the research efforts on flexible wireless communications. Our understanding of flexibility is the ability to manage whatever needs to be managed. The main focus of the initiative is identifying the necessary management elements for future wireless communications and incorporating them within evolving wireless systems.

One important part of the FlexNets initiative is the *flexible computing resource management project*. It stems from these studies and focuses on the development of a complete computing resource management framework for modern wireless communications. Other projects address the operating environment, hardware design, and waveform development [143].

The joint resource management concept of this dissertation will be concretized when merging advanced RRM concepts and contributions with the computing and application resource management of SDR platforms and applications. The result may be a cognitive engine that implements the joint resource management proposal of Chapter 8. This engine may initially manage those resources that facilitate the wireless access, although it should be aimed at the management of the entire resource pool in an end-to-end approach. The concepts of this dissertation may then also be practical for cognitive networks.

References

[1] R. Berezdivin, R. Breining, R. Topp, "Next-generation wireless communications concepts and technologies," *IEEE Commun. Mag.*, vol. 40, iss. 3, pp. 108-116, March 2002.

[2] 3rd Generation Partnership Project (3GPP) Web Site, www.3gpp.com

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

[4] J. Mitola III, "Software radios survey, critical evaluation and future directions," *IEEE AES Systems Magazine*, vol. 8, iss. 4, pp. 25-36, April 1993.

[5] J. Mitola III, "Cognitive radio: agent-based control of software radios", *Proc. 1st Karlsruhe Workshop on Software Radio*, Karlsruhe, Germany, March 2000.

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

[7] R. Baines, "The DSP bottleneck," *IEEE Commun. Mag.*, vol. 33, iss. 5, pp. 46-54, May 1995.

[8] J. Mitola III, "Technical Challenges in the globalization of software radio," *IEEE Commun. Mag.*, vol. 37, iss. 2, pp. 84-89, Feb. 1999.

[9] A. K. Salkintzis, Hong Nie, P. T. Mathiopoulos, "ADC and DSP challenges in the development of software radio base stations," *IEEE Pers. Commun.*, vol. 6, iss. 4, pp. 47-55, Auf. 1999.

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

[11] K. Lange, G. Blanke, R. Rifaat, "A software solution for chip rate processing in CDMA wireless infrastructure," *IEEE Commun. Mag.*, vol. 40, iss. 2, pp. 163-167, Feb. 2002.

[12] J. Mitola III, V. Bose, B. M. Leiner, T. Turletti, D. Tennenhouse (Eds.), *Software Radios*, *IEEE JSAC*, vol. 17, no. 4, pp. 509-747, April 1999.

[13] J. Mitola, Z. Zvonar, *Software Radio Technologies: Selected Readings*. IEEE Press, 2001.

[14] J. H. Reed, *Software Radio – A Modern Approach To Radio Engineering*, 2002 Prentice Hall PTR, Upper Saddle River, NJ.

[15] Software-Defined Radio (SDR) Forum Web Site, www.sdrforum.org

[16] A. Wiesler, J. K. Jondral, "A software radio for second- and third-generation mobile systems," *IEEE Trans. Vehic. Tech.*, vol. 51, iss. 4, pp. 738-748, July 2002.

[17] H. Lee et al., "Software defined radio – a high performance embedded challenge," *Proc. HiPEAC 2005*, *LNCS 3793*, pp. 6-26, 2005, Springer-Verlag Berlin Heidelberg 2005.

[18] R. Baines, D. Pulley, "A total cost approach to evaluating different reconfigurable architectures for baseband processing in wireless receivers," *IEEE Commun. Mag.*, vol. 41, iss. 1, pp. 105-113, Jan. 2003.

[19] R. Kokozinski, D. Greifendorf, J. Stammen, P. Jung, "The evolution of hardware platforms for mobile 'software defined radio' terminals," *Proc. 13th IEEE Int. Symp. Personal, Indoor and Mobile Radio Communications (PIMRC 2002)*, Lisbon, Portugal, 15-18 Sept. 2002, pp. 2389-2393 (vol. 5).

[20] A. Silberschatz, P. B. Galvin, G. Gagne, *Operating System Concepts*. John Wiley & Sons, Inc. 6th Edition, 2003.
[21] S. Vinoski, "CORBA: integrating diverse applications within distributed heterogeneous environments," *IEEE Commun. Mag.*, vol. 35, iss. 2, pp. 45-55, Feb. 1997.
[22] I. Foster, C. Kesselman (Eds.), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Mateo, CA, 1999.
[23] M. Parashar, C. A. Lee (Eds.), "Special issue on grid computing," *IEEE Proc.*, vol 93, iss. 3, March 2005.
[24] L. Smarr, C. E. Catlett, "Metacomputing," *Communications of the ACM*, vol. 35, no. 6, pp. 45-52, June 1992.
[25] A. Duller, G. Panesar, D. Towner, "Parallel Processing ─ the picoChip way!," *Proc. Communicating Process Architectures (CPA)*, 7-10 Sept. 2003, pp. 299-312.
[26] A. Jantsch, H. Tenhunen (Eds.), *Networks on Chip*. Kluwer Academic Publishers, 2003.
[27] T. Kogel, H. Meyr, "Heterogeneous MP-SoC - the solution to energy-efficient signal processing,", *Proc. 41st ACM IEEE Design Automation Conference (DAC'04)*, 7-11 June 2004, pp. 686-691.
[28] J. Villasenor, B. Hutchings, "The flexibility of configurable computing," *IEEE Signal Processing Mag.*, pp. 67-84, Sept. 1998.
[29] A. Gathener et al., "DSP-based architectures for mobile communications: past, present and future," *IEEE Commun. Mag.*, vol. 38, pp. 84-90, Jan. 2000.
[30] M. Cummings, S. Haruyama, "FPGA in the software radio," *IEEE Commun. Mag.*, vol. 37, pp. 108-112, Feb. 1999.
[31] picoChip Web Site, http://www.picochip.com/
[32] M. Kistler, M. Perrone, F. Petrini, "Cell multiprocessor communication network: built for speed," *IEEE Micro*, vol. 26, iss. 3, pp. 10-23, May-June 2006.
[33] A. Khokhar, V. K. Prasanna, M. Shaaban, C. L. Wang, "Heterogeneous computing: challenges and opportunities," *IEEE Computer*, vol. 26, iss. 6, pp. 18-27, June 1993.
[34] E. A. Lee, D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. C-36, No. 1, pp. 24-35, Jan. 1987.
[35] T. C. Hu, "Parallel sequencing and assembly line problem," *Oper. Res.*, vol. 9, pp. 841-848, Nov. 1961.
[36] T. L. Adam, K. M. Chandy, J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Comm. ACM*, vol. 17, no. 12, pp. 685-690, Dec. 1974.
[37] Y.-K. Kwok, I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406-471, Dec. 1999.
[38] 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.
[39] S. Selvakumar, C. S. R. Murthy, "Scheduling precedence constrained task graphs with non-negligible intertask communication onto multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 3, pp. 328-336, March 1994.
[40] G. C. Sih, E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 2, pp. 175-187, Feb. 1993.
[41] V. Chaudhary, J. K. Aggarwal, "A generalized scheme for mapping parallel algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, iss. 3, pp. 328-346, March 1993.
[42] Y.-K. Kwok, I. Ahmad, "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors, *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 5, pp. 506-521, May 1995.
[43] M. Tan et al., "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.
[44] I. Ahmad, Y.-K. Kwok, "On exploiting task duplication in parallel program scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 9, pp. 872-892, Sept. 1998.
[45] I. Ahmad, M. K. Dhodi, R. Ul-Mustafa, "DPS: dynamic priority scheduling heuristic for heterogeneous computing systems," *IEE Proc. Comput. Digit. Tech*., vol. 145, no. 6, Nov. 1998.

[46] T. Hagras, J. Janacek, "An approach to compile-time task scheduling in heterogeneous computing systems," *Proc. 2004 Int. Conf. Parallel Processing Workshops (ICPPW'04)*, 15-18 Aug., Montreal, Canada, pp. 182-189.

[47] C. Leangsuksun, J. Potter, "Design and experiments on heterogeneous mapping heuristics," *Proc. IEEE Heterogeneous Computing Workshop (HCW)*, pp. 17–22, April 1994.

[48] A. H. Alhusaini, V. K. Prasanna, C. S. Raghavendra, "A unified resource scheduling framework for heterogeneous computing environments," *Proc.8th Heterogeneous Computing Workshop (HCW'99)*, April 1999, pp. 156–165.

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

[50] A. H. Alhusaini, C. S. Raghavendra, V. K. Prasanna, "Run-time adaptation for grid environments," *Proc. IPDPS-01*, 23-27 April 2001, pp. 864-874.

[51] Y.-K. Kwok, A. A. Maciejewski, H. J. Siegel, A. Ghafoor, I. Ahmad, "Evaluation of a semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems," *Proc. Int. Symp. Parallel Architectures, Algorithms and Networks (ISPAN '99)*, 1999.

[52] K. Bondalapati, "Modeling and mapping for dynamically reconfigurable hybrid architectures," Ph.D. dissertation, University of Southern California, Aug. 2001.

[53] A.-R. Rhiemeier, F. Jondral, "Mathematical modeling of the software radio design problem," *IEICE Trans. Commun.*, vol. E86-B, no. 12, pp. 3456-3467, Dec. 2003.

[54] A.-R. Rhiemeier, "Modulares software defined radio," Ph.D. dissertation, Forschungsberichte aus dem Institut für Nachrichtentechnik der Universität Karlsruhe (TH), Band 9, Karlsruhe 2004.

[55] M.-Y. Wu, W. Shu, J. Gu, "Efficient local search for DAG scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 617-627, June 2001.

[56] H. Topcuoglu, S. Hariri, M.-Y. Wou, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, iss. 3, pp. 260-274, March 2002.

[57] S. Darbha, D. P. Agrawal, "Optimal scheduling algorithm for distributed-memory machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 1, pp. 87-95, Jan. 1998.

[58] R. Bajaj, D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 2, pp. 107-118, Feb. 2004.

[59] I. Ahmad, Y.-K. Kwok, "On parallelizing the multiprocessor scheduling problem," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, no. 4, pp. 414-432, April 1999.

[60] A. Dogan, F. Özgüner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, iss. 3, pp. 308-323, March 2002.

[61] S. Bansal. P. Kumar, K. Singh, "An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 14, no. 6, pp. 533-544, June 2003.

[62] D.-T. Peng, K. G. Shin, 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.

[63] C.-J. Hou, K. G. Shin, "Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems," *IEEE Trans. Comput.*, vol. 46., no. 12, pp. 1338-1356, Dec. 1997.

[64] K. Ramamritham, J. A. Stankovic, P.-F. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems," *IEEE Trans. Parallel Distr. Syst.*, vol. 1, no. 2, pp. 184-194, April 1990.

[65] K. Ramamritham, J. A. Stankovic, W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Trans. Comput.*, vol. 38, no. 8, pp. 1110-1123, Aug. 1989.

[66] D. Rosu, K. Schwan, S. Yalamanchili, R. Jha, "On adaptive resource allocation for complex real-time applications," *Proc. 18th IEEE Int. Real-Time Systems Symp.*, 2-5 Dec. 1997, pp. 320-329.

[67] K. Ecker et al., "An optimization framework for dynamic, distributed real-time systems," *Proc. 17th Int. Parallel and Distributed Processing Symp. (IPDPS 2003)*, Nice, France, 22-26 April 2003, IEEE CS Press.

[68] T. Xie, X. Qin, "Security-aware resource allocation for real-time parallel jobs on homogeneous and heterogeneous clusters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 5, pp. 682-697, May 2008.
[69] S. Gertphol, Y. Yu, A. Alhusaini, V. K. Prasanna, "A metric and mixed-integer-programming-based approach for resource allocation in dynamic real-time systems," *Proc. IPDPS-02*, 15-19 April 2002, pp. 993-1000.
[70] A. W. Krings, M. H. Azadmanesh, "Resource reclaiming in hard real-time systems with static and dynamic workloads" *Proc. 30th IEEE Hawaii Int. Conf. System Sciences (HICSS)*, 7-10 Jan. 1997, pp. 116-625.
[71] O. Moreira, J.-D. Mol, M. Beckooij, J. van Meerbergen, "Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix" *Proc. 11th IEEE Real Time Embedded Technology and Applications Symp. (RTAS'05)*, 7-10 March 2005, pp. 332-341.
[72] S. Stuijk, "Predictable mapping of streaming applications on multiprocessors," Ph.D. dissertation, TU Eindhoven, 2007.
[73] S. Stuijk, T. Basten, M. C. W. Geilen, H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs" *Proc. 44th ACM IEEE Design Automation Conf. (DAC 2007)*, 4-8 June 2007, pp. 777–782.
[74] S. Stuijk, M. Geilen, T. Basten, "Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs," *IEEE Trans. Comput.*, vol. 57, iss. 10, pp. 1331-1345, Oct. 2008.
[75] S. Stuijk, T. Basten, M. Geilen, A. H. Ghamarian, B. Theelen, "Resource-efficient routing and scheduling of time-constrained streaming communication on networks-on-chip", *Journal of Systems Architecture*, vol. 54, iss. 3-4, pp. 411-426, March-April 2008, Elsevier.
[76] J.-K. Kim et al., "Collective value of QoS: a performance measure framework for distributed heterogeneous networks," *Proc. IPDPS-01*, 23-27 April 2001, pp. 810–823.
[77] N. D. Doulamis, A. D. Doulamis, E. A. Varvarigos, T. A. Varvarigou, "Fair scheduling algorithms in grids," *IEEE. Trans. Parallel Distr. Syst.*, vol. 18, no. 11, pp. 1630-1648, Nov. 2007.
[78] H. Casanova, Y. Robert, H. J. Siegel (Eds.), *Algorithm design and scheduling techniques (realistic platform models) for heterogeneous clusters, IEEE Trans. Parallel Distr. Syst.*, *Special Section*, vol. 17, no. 2, pp. 97-190, Feb. 2006.
[79] A. Radulescu, A. J. C. van Gemud, "Low-cost task scheduling for distributed-memory machines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, iss. 6, pp. 648–658, June 2002.
[80] M.-Y. Wu, D. D. Gajski, "Hypertool: a programming aid for message-passing systems," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 1, no. 7, pp. 330-343, July 1990.
[81] Y. C. Lee, A. Y. Zomaya, "A novel state transition method for metaheuristic-based scheduling in heterogeneous computing systems," *IEEE Trans. Parallel Distr. Syst.*, vol. 19, no. 9, pp. 1215-1223, Sept. 2008.
[82] J. D. Ullman, "NP-complete scheduling problems." *J. Comput. System Sciences*, vol. 10, pp. 384-393, 1975.
[83] R. Mehrotra, S. N. Talukdar, "Scheduling of tasks for distributed processors," *Proc. 11th Int. Symp. Computer Architecture (ISCA'84)*, Jan. 1984, pp. 263-270.
[84] S. H. Bokhari, "On the mapping problem," IEEE Trans. Comput., vol. C-30, no. 3, pp. 207-214, March 1981.
[85] J. A. Stankovic, M. Di Natale, G. C. Buttazzo, "Implications of classical scheduling results for real-time systems," *IEEE Computer*, vol. 28, iss. 6, pp. 16-25, June 1995.
[86] J. Lee, S. Kim, J. Park, "Q-SCA: Incorporating QoS support into software communications architecture for SDR waveform processing," *Real-Time Syst (2006)*, pp. 19-35, Springer Science + Business Media, LLC 2006.
[87] JTRS Software Communications Architecture (SCA) Web Site, http://sca.jpeojtrs.mil/
[88] Modular Software-programmable Radio Consortium, "Software communications architecture specification," *Tech. Rep.* MSRC-5000SCA, v.2.2, 17 Nov. 2001.
[89] Open Source SCA Implementation Embedded (OSSIE) Web Site, http://ossie.wireless.vt.edu/trac
[90] J. Bertrand, J. W. Cruz, B. Majkrzak, T. Rossano, "CORBA delays in a software-defined radio", *IEEE Commun. Mag.*, vol. 40, iss. 2, pp. 152-155, Feb. 2002.
[91] D. Oldham, M. Scardelleti, "JTRS/SCA and custom/SDR waveform comparison", *Proc. 2007 Military Communications Conf. (MILCOM 2007)*, Orlando, FL, 29-31 Oct. 2007.

[92] I. Gomez, V. Marojevic, J. Salazar, A. Gelonch, "A lightweight operating environment for next generation cognitive radios," *Proc. 11th Euromicro Conf. Digital Systems Design (DSD 2008)*, Parma, Italy, 3-5 Sept. 2008.
[93] S. Kim, J. Masse, S. Hong, "Dynamic deployment of software-defined radio components for mobile wireless internet applications," *Proc. 2nd Int. Conf. Human Society@Internet*, Seoul, Korea, June 18-20, LNCS 2713/2003, pp. 694-700, Springer Berlin / Heidelberg, 2003.
[94] Ismael Gomez, "A software framework for software radio," Master Thesis, Dept. Signal Theory and Communications, Polytechnic University of Catalonia (UPC), Jan. 2008.
[95] S.-L. Tsao, C.-C. Lin, C.-L. Chiu, H.-L. Chou, M.-C. Wang, "Design and implementation of software framework for software defined radio system," *Proc. IEEE 56th Vehicular Technology Conf. (VTC 2002-Fall)*, 24-28 Sept. 2002, pp 2395-2399 (vol. 4).
[96] G. J. Minden et al., "An agile radio for wireless innovation," *IEEE Commun. Mag.*, vol. 45, iss. 5, pp. 113-121, May 2007.
[97] The GNU software radio (GNU Radio) Web Site, http://gnuradio.org/trac
[98] M. Palkovic, H. Cappelle, M. Glassee, B. Bougard, L. Van der Perre, "Mapping of 40 MHz MIMO SDM-OFDM baseband Processing on Multi-Processor SDR Platform," *Proc 11th IEEE Workshop Design and Diagnostics of Electronic Circuits and Systems (DDECS 2008)*, Bratislava, Slovakia, 16-18 April 2008.
[99] Q. Zhang, A. B. J. Kokkeler, G. J. M. Smit, "A system-level design method for cognitive radio on a reconfigurable multi-processor architecture," *Proc. Int. Symp. System-on-Chip 2007 (SOC 2007)*, Tampere, Finland, 20-21 Nov. 2007.
[100] B. Mohebbi, E. C. Filho, R. Maestre, M. Davies, F. J. Kurdahi, "A case study of mapping a software-defined radio (SDR) application on a reconfigurable DSP core," *Proc. 1st IEEE/ACM/IFIP Int. Conf. Hardware/Software Codesign and System Synthesis (CODES+ISSS 2003)*, Newport Beach, CA, 1-3 October 2003.
[101] G. K. Rauwerda, P. M. Heysters, G. J. M. Smit, "Mapping wireless communication algorithms onto a reconfigurable architecture," *The J. of Supercomputing*, vol. 30, pp. 263-282, 2004 Kluwer Academic Publishers.
[102] J. Park, S. Ha, "Performance analysis of parallel execution of H.264 encoder on the cell processor," *Proc. IEEE/ACM/IFIP Workshop Embedded Systems for Real-Time Multimedia (ESTIMedia 2007)*, 4-5 Oct. 2007, pp. 27-32.
[103] X. Reves, A. Gelonch, V. Marojevic, 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.
[104] D. F. Robinson, L. R. Foulds, *Digraphs: Theory and Techniques*. Gordon and Breach Science Publisher Inc., 1980.
[105] J. A. Bondy, U. S. R. Murty, *Graph Theory with Applications*. Elsevier Science Publishing Co., Inc., New York, Fifth Printing, 1982.
[106] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1974.
[107] R. Tanner, J. Woodard, *WCDMA – Requirements and Practical Design*. John Wiley and Sons Inc., 2004.
[108] Technical Specification Group Radio Access Network (3GPP), "TS 25.212 V6.4.0 – multiplexing and channel coding (FDD)," March 2005, www.3gpp.org
[109] Technical Specification Group Radio Access Network (3GPP), "TS 25.213 V5.5.0 – spreading and modulation (FDD)," Dec. 2003, www.3gpp.org
[110] T. Faber, M. Schönle, "DSP-platform target report," SLATS Consortium, Project no. 27016, Deliverable D23, Dec. 1999.
[111] Texas Instruments (TI) Web Site, www.ti.com
[112] J. M. Pereira, "Beyond software radio, towards reconfigurability across the whole system and across networks," *Proc. IEEE 50th Vehicular Technology Conf. (VTC 1999-Fall)*, pp. 2815-2818.
[113] R. W. Thomas, D. H. Friend, L. A. DaSilva, A. B. MacKenzie, "Cognitive networks: adaptation and learning to achieve end-to-end performance objectives," *IEEE Commun. Mag.*, vol. 44, iss. 12, pp. 51-57, Dec. 2006.

[114] J. Mitola, G. Q. Maguire, "Cognitive radio: making software radios more personal," *IEEE Pers. Commun.*, vol. 6, no. 4, pp. 13-18, 1999.
[115] J. Mitola, "Cognitive radio: an integrated agent architecture for software defined radio," Ph.D. dissertation, Royal Institute of Technology (KTH), Stockholm, Sweden, May 2000.
[116] S. Haykin, "Cognitive radio: brain-empowered wireless communications," *IEEE JSAC*, vol. 23, no. 2, pp. 201-220, Feb. 2005.
[117] J. Mitola III, "Software radio architecture evolution: foundations, technology tradeoffs, and architecture implications," *IEICE Trans. Commun.*, vol. E83-B, no. 6 (*Special Issue on Software Defined Radio and its Technologies*), pp. 1165-1173, June 2000.
[118] D. Maldonado et al., "Cognitive radio applications to dynamic spectrum allocation," *Proc. 1st IEEE Int. Symp. New Frontiers in Dynamic Spectrum Access Networks (DYSPAN 2005)*, Baltimore, MD, 8-11 Nov. 2005, pp. 597-600.
[119] F. K. Jondral, "Software-defined radio–basics and evolution to cognitive radio," *EURASIP J. Wireless Communications and Networking*, vol. 2005, no. 3, pp. 275-283, 2005.
[120] R. J. Mayher, F. Wentland, "Spectrum management structure and regulations in the US – Do we need a change in the 21st century?", *Proc. 1990 IEEE Int. Symp. Electromagnetic Compatibility*, 21-23 Aug. 1990, pp. 380-385.
[121] G. Gerrard, "The regulation of mobile communications," *Proc. IEEE 6th Int. Conf. Mobile Radio and Personal Communications*, 9-11 Dec. 1991, pp. 131-135.
[122] J. Pérez-Romero, O. Sallent, R. Agustí, M. A. Diaz-Guerra, *Radio Resource Management Strategies in UMTS*. John Wiley & Sons, 2005.
[123] J. Nasreddine O. Sallent J. Pérez-Romero R. Agustí, "Advanced spectrum management in wideband code division multiple access systems enabling cognitive radio usage," *IET Communications* (Special Issue on Cognitive Spectrum Access), vol. 2, iss. 6, pp. 794-805, July 2008.
[124] P. Demestichas, G. Vivier, K. El-Khazen, M. Theologou, "Evolution in wireless systems management concepts: from composite radio environments to reconfigurability," *IEEE Commun. Mag.*, vol. 42, iss. 5, pp. 90-98, May 2004.
[125] J. Hoffmeyer, I.-P. Park, M. Majamundar, S. Blust, "Radio software download for commercial wireless reconfigurable devices," *IEEE Commun. Mag*., vol. 42, iss. 3, pp. 26-32, March 2004.
[126] P. Leaves et al., "Dynamic spectrum allocation in composite reconfigurable wireless networks," *IEEE Commun. Mag.*, vol. 42, iss. 5, pp. 72-81, May 2004.
[127] I. Katzela, M. Naghshineh, "Channel assignment schemes for cellular mobile telecommunication systems: a comprehensive survey," *IEEE Pers. Commun.*, vol. 3, iss. 3, pp. 10-31, June 1996.
[128] F. Box, "A heuristic technique for assigning frequencies to mobile radio nets," *IEEE Trans. Vehic. Tech.*, vol. 27, iss. 2, pp. 57-64, May 1978.
[129] J. M. Peha, "Wireless communications and coexistence for smart environments," *IEEE Pers. Commun.*, vol. 7, iss. 5, pp. 66-68, Oct. 2000.
[130] Y. Xing, R. Chandramouli, S. Manglold, S. Shanakar N, "Dynamic spectrum access in open spectrum wireless networks," *IEEE JSAC*, vol. 24, no. 3, pp 626-637, March 2006.
[131] T. A. Weiss, F. K. Jondral, "Spectrum pooling: an innovative strategy for the enhancement of spectrum efficiency," *IEEE Commun. Mag.*, vol. 42, iss. 3, pp. S8-S14, March 2004.
[132] J. Perez-Romero et al., "Common radio resource management: functional models and implementation requirements," *Proc. PIMRC 2005*, Berlin, 11-14 Sept. 2005, pp. 2067-2071.
[133] Q. Zhao, B. M. Sadler, "A survey of dynamic spectrum access," *IEEE Signal Processing Mag.*, vol. 24, iss. 3, pp. 79-89, May 2007.
[134] D. N. Hatfield, P. J. Weiser, "Property rights in spectrum: taking the next step," *Proc. 2005 1st IEEE Int. Symp. New Frontiers in Dynamic Spectrum Access Networks (DySPAN 2005)*, 8-11 Nov. 2005, pp.43-55.
[135] L. Xu, "DRiVE-ing to the internet: dynamic radio for IP services in vehicular environments," *Proc. 25th Annual IEEE Conf. Local Computer Networks (LCN 2000)*, 8-10 Nov. 2000, pp 281-289.
[136] W. Lehr, J. Crowcroft, "Managing shared access to a spectrum commons," *Proc. 2005 1st IEEE Int. Symp. New Frontiers in Dynamic Spectrum Access Networks (DySPAN 2005)*, 8-11 Nov. 2005, pp.420-444.
[137] J. Mitola, "Cognitive radio for flexible mobile multimedia communications," *Proc. IEEE Int. Workshop Mobile Multimedia Communications* (*MoMuC'99*), 15-17 Nov. 1999, pp. 3-10.

[138] L. Giupponi et al., "Towards balancing user satisfaction and operator revenue in beyond 3G cognitive networks," *Proc. 15th IST Mobile and Wireless Summit*, Myconos, 4-8 June 2006.
[139] R. van Nee, R. Prasad, "OFDM for Wireless Multimedia Communications," Artech House, London, 2000.
[140] E. M. Noam, "Taking the next step beyond spectrum auctions: open spectrum access", *IEEE Commun. Mag.*, vol. 33, iss. 12, pp. 66-73, Dec. 1995.
[141] D. Marsh, "Software-defined radio tunes in," *Electronic Design, Strategy, News (EDN)*, March 3, 2005, pp. 52-63, available at http://www.edn.com/article/CA505082.html.
[142] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, S. Mohanty, "NeXt generation/dynamic spectrum access/cognitive radio wireless networks: a survey," *Elsevier Computer Networks* 50 (2006), pp. 2127-2159.
[143] Flexible Wireless Communications Systems and Networks (FlexNets) Web Site, http://flexnets.upc.edu/trac
[144] V. Marojevic, X. Revés, A. Gelonch, "An open computing resource management framework for real-time computing," *Proc. 15th Int. Conf. High Performance Computing (HiPC 2008)*, Bangalore, India, 17-20 Dec. 2008, Lecture Notes in Computer Science 5374, pp. 169-182, Springer Berlin Heidelberg.
[145] C. C. Ribeiro, S. L. Martins, I. Rosseti, "Metaheuristic for optimization problems in computer communications," *Elsevier Computer Communications* 30 (2007), pp. 656-669.
[146] D. E. Kirk, "An Introduction to dynamic programming," *IEEE Trans. Education*, vol. 10, iss. 4, pp. 212-219, Dec. 1967.