EURASIP Journal on Wireless Communications and Networking 2005:3, 333–342

Modular Software-Defined Radio

Arnd-Ragnar Rhiemeier

Institut für Nachrichtentechnik, Universität Karlsruhe (TH), 76128 Karlsruhe, Germany
Email: rhiemeier@int.uni-karlsruhe.de

Received 1 October 2004; Revised 14 February 2005

In view of the technical and commercial boundary conditions for software-defined radio (SDR), it is suggestive to reconsider the concept anew from an unconventional point of view. The organizational principles of signal processing (rather than the signal processing algorithms themselves) are the main focus of this work on modular software-defined radio. Modularity and flexibility are just two key characteristics of the SDR environment which extend smoothly into the modeling of hardware and software. In particular, the proposed model of signal processing software includes irregular, connected, directed, acyclic graphs with random node weights and random edges. Several approaches for mapping such software to a given hardware are discussed. Taking into account previous findings as well as new results from system simulations presented here, the paper finally concludes with the utility of pipelining as a general design guideline for modular software-defined radio.

Keywords and phrases: flexible digital baseband signal processing, firmware support for reconfiguration, computing resource allocation, multiprocessing, modeling of SDR software.

1. INTRODUCTION

Software-defined and hardware reconfigurable radio systems have attracted more and more attention recently because they are expected to be among the key techniques to serving future wireless communication market needs. In contrast to the strong convergence tendency in wired networks, a growing number of standards and communication modes can be observed in wireless access networks. Presumably, this trend will prevail, eventually due to the natural diversity in service requirements and radio environments. The better the match between the physical channel (the properties of which are determined in part by the user mobility) and the signal processing in the transceiver, the easier to achieve the optimal quality of service (QoS) on the physical layer. Furthermore, the ongoing introduction of UMTS in Europe shows that diversity in standards is not only a technical challenge; if market response falls short of business expectations (based on a particular communication standard), or if user's demand shifts to a different wireless access technology (and thus to a different sort of underlying signal processing), it would be beneficial for any manufacturing company to be able to respond quickly to such situations. Software-defined and reconfigurable radio systems have the potential to allow short time-to-market product designs under these commercial conditions.

The present paper puts an emphasis on the physical-layer signal processing because its capabilities represent the fundamental limits for higher layers in delivering their services. Therefore, mastering *all* aspects of physical-layer signal processing in software-defined and hardware reconfigurable radios is vital for delivering best end-to-end service to the end user, by means of a single communication device. Modular software-defined radio (Mod-SDR) strives for casting light on one important aspect which has been largely neglected hitherto: the design guidelines which govern the coordinated interplay of signal processing software modules embedded in logical structures of some arbitrary wireless communication standards.

1.1. *Related work*

A great number of important contributions on software-defined radio [1, 2, 3] and reconfigurability [4, 5, 6, 7, 8] can be found in the literature. However, many authors narrow down their research interest to one particular aspect of the signal processing chain: sample rate adaptation [9, 10, 11], RF front-end design [12, 13, 14, 15, 16], A/D conversion [17, 18], or channel coding [19, 20, 21], just to name a few examples. Notably, work related to signal processing in the digital baseband is centered around algorithms [22, 23]. However, structural properties of signal processing software (including an abstract way for representation) and the principles of organizing the execution of multiple algorithms in a distributed multiprocessing hardware system have not been studied intensively in the context of software radio. One major contribution of Mitola [24] attempts to reexamine software radio from a truly unorthodox point of view, but his findings still pertain to algorithms and eventually do not reach beyond the Turing's theory of computing. Nevertheless, his contribution hints at the fact that SDR requires an understanding which is radically different from classical communications and its BER curves.

1.2. Motivation for modular software-defined radio

The motivation for introducing a novel view of flexible radio system design is twofold. First, the fact that a radio terminal accomplishes its signal processing by software or reconfigurable hardware rather than by dedicated hardware does not render the rules of real-time computing obsolete, but so far this aspect has not been considered systematically in the context of software-defined radio. Second, any BER produced by some software implementation has to be at least as good as the BER of its equivalent ASIC implementation. Hence, it appears to be questionable to present BER curves as a measure of quality for the design of a software-defined radio.

The goal of modular software-defined radio is to establish general guidelines for designing and operating flexible signal processing systems. In order to make these guidelines general, they need to be independent of particular communication standards, independent of algorithmic implementation details, and independent of technological advances in microelectronics. The model for SDR software will reflect these important aspects.

1.3. Compatibility with the real world

In the same way as the ideal software radio concept [1, 25] has evolved into some compromise approaches usually summarized under the term of software-defined radio, the advent of reconfigurable, distributed signal processing hardware in radio devices [26, 27] can be seen as another step in this evolution towards implementations which are both technologically feasible and economically attractive.

Critics generally claim that SDRs will always be notoriously power-inefficient and inherently overpriced, hence never prove competitive against carefully designed ASICs. This may be true indeed if the flexibility of SDR is unconditionally passed on to the end user in the form of "future upgradability." Therefore, it is more reasonable to predict that the flexibility of SDR is likely to stay under the immediate control of manufacturers, all the more so to support a sustainable business model. Actually, time-to-market is the master argument in favor of software-defined radio techniques. The present paper shares this view suggesting to perceive software-defined and reconfigurable radios as wireless communications embedded real-time systems which are tunable to end-user needs or network operator needs, but not more. Modular software-defined radio provides those embedded systems with design guidelines and the core of a QoS manager for the physical layer.

2. MODELING OF HARDWARE AND SOFTWARE

One fundamental assumption is that SDR will become reality soonest in the form of some distributed multiprocessing hardware architecture, providing sufficient computing power at a much lower electric power consumption than that of a single comparable general-purpose processor. Furthermore, all hardware resources of a Mod-SDR device such as processors, buses, memories, and interfaces are administered by a nonpreemptive operating system. Supplied by the terminal manufacturer, this piece of firmware is protected from any direct manipulation on the part of the end user. However, by means of a physical layer API, both user applications and the network may address transmission mode requests to the QoS manager which belongs to the core framework services/applications running immediately on top of the operating system (see SCA, [28, Figure 2-1]). A request will certainly include an abstract representation of the signal processing software which needs to be executed to realize the transmission mode. modular software-defined radio makes use of directed acyclic graphs to represent signal processing software.

The main task of the QoS manager consists of the mapping of signal processing software to the available hardware while respecting the real-time requirements which are defined by the air interface of the requested communication standard [29]. In return for requests, the QoS manager accepts or rejects transmission modes, based on the availability of resources and on a decision process which includes partitioning of the graph and scheduling of all software modules. Partitioning is the process of uniquely assigning each module to a processor, while scheduling means determining individual trigger instants for each module.

Throughout this work, a software module is defined to be the smallest entity of executable machine code, which cannot be preempted by the operating system. Software modules are self-contained, independent entities, and there is no data exchange across module boundaries other than input data from predecessor modules and output data to successor modules. In principle, any module may be connected to any other module, the only requirement being data type compatibility in between all modules.

2.1. Details of the software model

For the Mod-SDR software model to be independent of processor types (DSP, FPGA, ASSP, specific coprocessor, and ASIC) and of technological advances in microelectronics, the processing runtime of software modules is taken into account as the main behavioral attribute of signal processing software. At the same time, algorithmic details are abstracted away in this manner. Given the framework of directed acyclic graphs, the nodes of a graph represent software modules carrying some signal processing runtime as the node weight. Due to the vast variety of unpredictable influences on the processing runtime of software modules in an SDR environment [29], node weights are subject to random variation within a stochastic linear resource runtime model [30]. The basic assumption of linearity originates from the idea that the more processing runtime is needed, the more data is to be produced at the output of a software module:

$$p_m = \alpha \cdot c \cdot r_m, \quad \forall \text{ nodes } m : 1 \leq m \leq M, \tag{1}$$

where p_m is the processing runtime of some node m and r_m is m's output memory resource demand. Formally, $\alpha \in \mathbb{R}^+$ is the constant of proportionality translating a memory demand into a runtime, hence its unit $[\alpha] = (\text{bit/s})^{-1}$. It can be interpreted as the absolute speed of a processor when executing the signal processing machine code behind node m. The constant factor c is unitless, and its value is drawn from a random experiment, for each m anew. The characteristics of the SDR environment as well as all shortcomings of a strictly linear resource runtime model are modeled by the real-valued random variable C. A complete description of this variable is given by its probability density function (pdf) $f_C(c)$. Throughout this paper, all realizations c stem from independent identically distributed random variables C for all nodes m. The pdf employed in this work has a rectangularly windowed Gaussian shape with the parameters $\mu_C = 1.0$, $\sigma_{C,\text{eff}} = E\{(C - \mu_C)^2\} = 0.25$, and windowed interval $[0.5; 1.5]$. The choice of a Gaussian is certainly arbitrary, but there are strong hints that the actual shape of the pdf has much less influence on the performance of the QoS manager than the effective relative spread $\sigma_{P,\text{eff}}/\mu_{P,\text{eff}}$ of processing runtimes [31].

The structural properties of Mod-SDR software are captured in directed edges $\langle m, n\rangle$ between a pair of nodes m and n of a graph. In order to be independent of any particular communication standard, those graphs are not only random in their node runtimes, but also in their directed edges.

The random graph generator which is used for computer simulations produces irregular, connected, directed, acyclic graphs with a fixed number of nodes $M = 40$. The generation process starts out from a chain of two nodes indexed $m = 1$ (referred to as the source node) and $m = 2$ (referred to as the target node). Subsequently, nodes are added in an iterative process by placing them somewhere relative to the existing graph. For every node to be placed, one predecessor and one successor are selected at random and with equal probability from the existing nodes of the graph. Should a new node shortcut the edge between adjacent existing nodes, then that edge is removed with probability of 0.5. Connectedness of the graph is enforced in a simple way by exempting $m = 1$ from the node selection procedure, whereas the property of noncyclicity needs to be verified throughout the entire graph generation process. Finally, some additional graph properties, which are related to the data input and output behavior of nodes, need to be determined. First, the output-to-input data ratio of all nodes be 1.0. Second, all edges outgoing from any demultiplexing node convey the full output data volume. Third, all edges incoming to any multiplexing node convey only one Kth of the predecessors' output data volumes, where K is the number of incoming edges. These rules make sure that edge weights remain in the same order of magnitude throughout the entire graph. Figure 1 shows one realization of a random graph as an example.

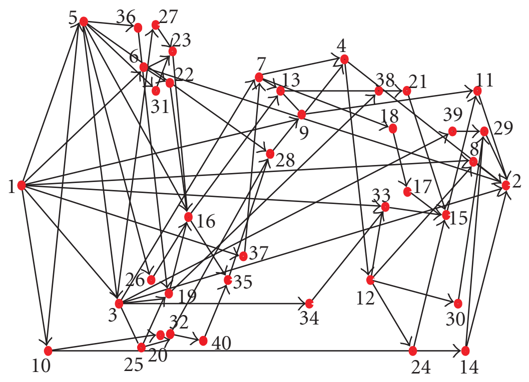

Figure 1: Irregular, connected, directed, acyclic graph.

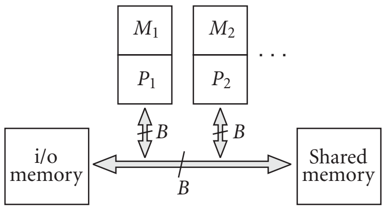

Figure 2: Symmetric multiprocessor architecture.

2.2. Details of the hardware model

A symmetric multiprocessor architecture (see Figure 2) serves as the model of a modular, easy-to-extend multiprocessing hardware system for Mod-SDR. The architecture generally includes $L \in \mathbb{N}$ identical processors P_l with associated (distributed) memory M_l, a shared memory and an input/output (i/o) memory for interfacing of the physical layer signal processing subsystem to the outside world, that is, to other processing subsystems or to the analog front end of the Mod-SDR transceiver.

All of these hardware resources are connected by $B \in \mathbb{N}$ separate data buses. It is assumed that processors are actively involved in bus transfers, that is, no useful signal processing code of a module can be executed by a processor while this processor is exchanging data with the shared memory or with the i/o memory via some bus. This is certainly a conservative assumption, which can also be interpreted as a worst case on one hand. On the other hand, however, it is very unlikely that simultaneous signal processing and bus communications can be achieved in the general case of Mod-SDR graphs. Even if a processor architecture supports communication latency hiding behind useful core computations, those mechanisms would require coordination on the code programing level across module boundaries. However, this intermodule control flow contradicts the Mod-SDR self-containment paradigm, where any module may indeed be linked to any other module (logically, by the directed edge of a graph), in order to accomplish a useful signal processing task, but without mutual knowledge of their respective machine code internals.

Naturally, bus access is exclusive. Conflicting access timings on the part of processors need to be arbitrated by the scheduler, which is built into the QoS manager. The bus speed is given relative to the signal processing speed α of the processors, in the form of a relative bus speed $\beta \in \mathbb{R}^+$. The unitless factor β describes how much faster a processor can transfer an amount of data over the bus rather than produce the same amount of data as the output of a module (by regular signal processing). Large values of the relative bus speed ($\beta \gg 1$) represent fast buses, whereas $\beta < 1$ represents slow buses. Partitioning always entails a cut affecting a certain subset of edges in the graph. The logical data flow along cut edges then translates into an asynchronous, physical data flow between processors via the shared memory. How this is organized using pairs of bus transfer nodes is described in detail in [32]. The basic idea is that bus transfers require runtime in the same way as regular signal processing nodes, but paired for intermediate results to be written to and read from the shared memory. Therefore, the resulting runtime model for bus transfer nodes is similar to (1), but includes β in the place of c:

$$p_{\langle m,n \rangle} = \alpha \cdot \beta^{-1} \cdot r_m, \quad \forall \text{ edges } \langle m,n \rangle \text{ affected by the cut.} \tag{2}$$

These runtimes do not depend on the SDR environment, but on the deterministic capabilities of the Mod-SDR device. Their values $p_{\langle m,n \rangle}$ reappear later as edge weights that represent potential link cost for all edges $\langle m,n \rangle$. Actual costs are only incurred by those edges affected by the partitioning cut.

The focus of the present work is on fundamental design and operating principles in Mod-SDR systems. Before tackling the general case of $L \in \mathbb{N}$ processors, the case $L = 2$ should be well understood first. With $L = 2$ processors, only $B \leq 2$ is reasonable. In this paper, the case of $B = 1$ bus is studied.

2.3. Proposed measure of quality

It is the declared goal of Mod-SDR to go beyond designing for some particular set of standards or transmission modes. Therefore, concrete real-time requirements such as deadline periods (which are clearly determined by the air interfaces of any standard) are missing. Instead, the more general requirement for maximum *speedup* of a multiprocessing hardware system is considered. The speedup $s \in \mathbb{R}^+$ is defined as the factor by which the multiprocessor Mod-SDR implementation terminates faster than the same software implementation on a single processor running at the same speed of α. The advantage of speedup is evident; it is a relative measure of quality for the design of (and the way of operating) a multiprocessor computing system. The actual value of α is not of importance because it cancels out in the speedup s.

3. MAPPING APPROACHES

Although the Mod-SDR software model includes random graphs, statistics in this approach should not be confused with any "statistical structure of computational demand" [24]. The way of building and using *a population* of SDR terminals involves a random process, but transmission mode requests are *realizations* of this random process. Therefore, the QoS manager has to deal with *realizations* of random graphs. Per request, the total number of nodes and all associated node weights are arbitrary, but fixed. Likewise, the logical structures captured in the set of directed edges remain fixed over their entire utilization period. Potentially, QoS parameters may be negotiated during connection setup, but once a transmission mode has been accepted, the (static) scheduling situation is deterministic, and so is the demand for computing power [29].

All SDR design techniques related to mapping structured signal processing software to hardware may be roughly classified by the number of graph copies involved in the partitioning and scheduling process; some approaches use but a single copy, while others imply multiple identical copies of the graph. In principle, techniques of both classes are applicable to signal processing for circuit-switched services as well as for packet-switched services. However, ordinary protocol requirements in packet-oriented networks (e.g., stop-and-wait ARQ) may oftentimes force the QoS manager to operate on a single copy of the given graph. As a matter of principle, single-copy variants are less demanding in terms of program memory and data memory.

The following partitioning approaches are employed in Mod-SDR system simulations to be discussed in a later section of the this paper.

(i) Implicit partitioning by the *Hu* algorithm [29]—eventually, a scheduling algorithm.

(ii) Kernighan-Lin (*KL*) algorithm [30]—a method for local search of the design space.

(iii) *Spectral* partitioning [33]—an application of algebraic graph theory.

These approaches primarily operate on a single copy of the graph which is given by some transmission mode request. The originally implied scheduling idea is that radio signals are processed on a frame-by-frame basis, only one frame per real-time period, and all computations relate to a single radio frame only. An accurate pseudocode of the static scheduler is given in [34].

However, pipelined scheduling [33, 35] in combination with these partitioning approaches revokes the memory advantage of a single graph copy, because pipelining involves the processing of radio signals related to several different radio frames within the time of one real-time period. The number of different radio frames involved in these computations is called the depth of the pipeline. Indeed, the program memory remains unaffected by pipelining, but not the data memory; multiple intermediate computation results from several different frames must be kept in memory, which results in a much higher demand than before.

A radical alternative to the above partitioning approaches is graph duplication; one complete copy of the graph is assigned to the first processor, another copy is assigned to the second processor. No partitioning algorithm is needed, the workload on both processors is perfectly balanced by construction, and bus access is reduced to the necessary i/o data transfers. The associated scheduling scheme has been named graph duplication pipelining (*GDP*) [35]. Despite its obvious advantages, GDP suffers from both increased data memory and program's memory demand due to the duplication process. An alternative pipelining approach that returns to operating on a single copy of the graph is half-frame pipelining (*HFP*) [36, 37]. HFP is primarily based on a scheduling

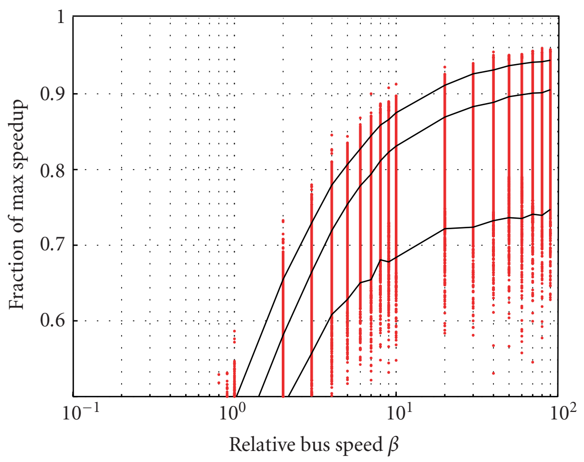

FIGURE 3: Implicit Hu partitioning, no pipelining.

idea where the QoS manager supports the following kind of time-interleaved frame processing; while one processor is still busy processing the second half of some frame indexed i, the other processor already starts processing its successor frame indexed $(i+1)$. In contrast to GDP, this approach requires load-balanced partitioning of the graph under one additional side condition, namely maintaining a vertical cut relative to the source node and the target node. Both the KL algorithm and a modified variant of spectral partitioning have been tested for this purpose. It can be shown [38] that the KL algorithm is a simple and efficient partitioning support for HFP.

In principle, all of these mapping approaches are eligible for application by the QoS manager of a Mod-SDR, independent of the service type. It remains to be shown which approach achieves a high probability of good speedup in the highly unpredictable SDR environment.

4. SYSTEM SIMULATIONS OF MOD-SDR

The three partitioning approaches have been discussed extensively elsewhere, but the comparison of performance was merely based on one particular sample graph. The node weights did stem from random experiments, but the structure of the graph remained fixed throughout all simulation runs. Here, in contrast, the random graph model of Section 2.1 (also including random edges) is applied to improve the expressiveness of Mod-SDR system simulations. Those new results are discussed in the following.

All figures show speedup measurements (dots) as a function of the relative bus speed β. These results are given as a fraction of the maximum speedup $\bar{s}$ which is theoretically achievable [33] by perfect parallelization. Hence, on one hand, a fractional value of 1.0 represents the upper performance limit for any Mod-SDR realization. On the other hand, the fractional value of 0.5 represents a reasonable lower limit, because below $s = 0.5$, the two-processor system would effectively work slower than a single-processor system, thus rendering any distributed processing approach meaningless in principle. At a sample size of 2000 realizations per β, the observed measurement range is often so densely populated by dots that the latter amalgamate into vertical lines. Therefore, in addition to the individual speedup measurements, the contour lines of the 5%, the 50% (median), and the 95% quantile are estimated and overlayed to the figures. A contour line represents the maximum speedup achieved by the given quantile of Mod-SDR realizations (95%: topmost, 5%: bottommost, median: in between) as a function of β.

4.1. *Circuit-switched services*

Figure 3 shows the speedup results for implicit Hu partitioning and nonpipelined scheduling. Obviously, the faster the bus, the better the speedup. Since Hu's algorithm is eventually a pure scheduling algorithm, it does not take into account any link cost while partitioning graphs implicitly. A naive working assumption could be that speedup approaches the limit of 1.0, if the bus is somewhat fast enough, because link cost approaches zero as $\beta \rightarrow \infty$. Figure 3 disproves this assumption. The behavior observed over the entire β range can be explained in part by the occurrence of PHYSICAL_WAIT_IDLE and LOGICAL_WAIT_IDLE conditions [30]. The former arises after the arbitration of concurrent bus access requests, whereas the latter originates from logical interdependencies of nodes in the graph; although the bus is fully accessible, one processor is forced to remain idle waiting for some intermediate results to be produced by the other processor.

Both conditions cause idle times in the processors, and thus reduce speedup. While concurrent bus access requests become more and more unlikely as bus speed increases, the LOGICAL_WAIT_IDLE condition continues to prevail in all schedules independent of β. Another reason for speedup loss against the limit can be identified in a special property of Hu's algorithm; the approach strictly aims at maximum parallelization. However, the random graph model produces realizations with a degree of inherent parallelism $\tilde{d}$, $0.3 \leq \tilde{d} \leq 0.8$ [36]. As a consequence, if the graph cannot be parallelized due to its given structural properties (small $\tilde{d}$ value below $\tilde{d} = 0.5$), the Hu algorithm systematically fails to generate high speedup. Load imbalance between the two processors (equal to the difference of aggregate runtimes between the two partitions) is the resulting effect of this failure.

Pipelining eliminates LOGICAL_WAIT_IDLE conditions by deliberately constructing a dense schedule in a first step. All resulting anticausal data dependencies between the partitions are resolved in a second step by rescheduling bus transfers of intermediate results across the boundaries of real-time processing periods. In this way, a radio frame pipeline of some depth is created (cf. Section 3). Figure 4 shows the speedup results for implicit Hu partitioning under pipelining. Indeed, the overall speedup behavior has improved; all contour lines indicate higher speedup for the same quantile of realizations, and the speedup spread for fast buses is reduced. Nevertheless, implicit partitioning pursuant to Hu's algorithm continues to suffer from its systematic drawbacks mentioned above: strong dependency on graph structure and complete insensitivity to link cost.

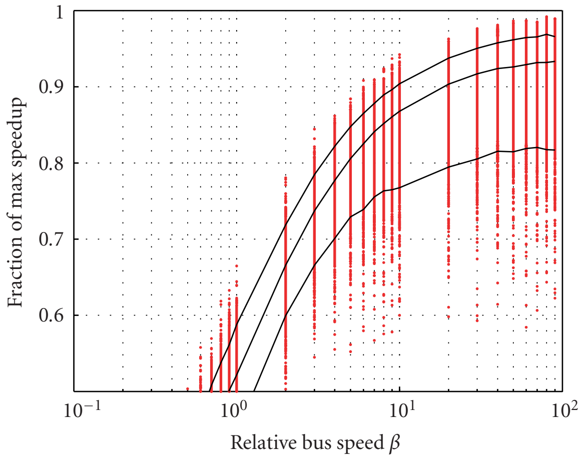

FIGURE 4: Implicit Hu partitioning, pipelining.

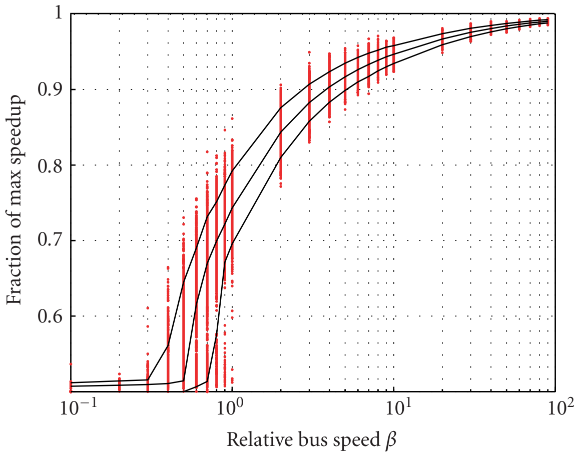

FIGURE 6: KL partitioning, pipelining.

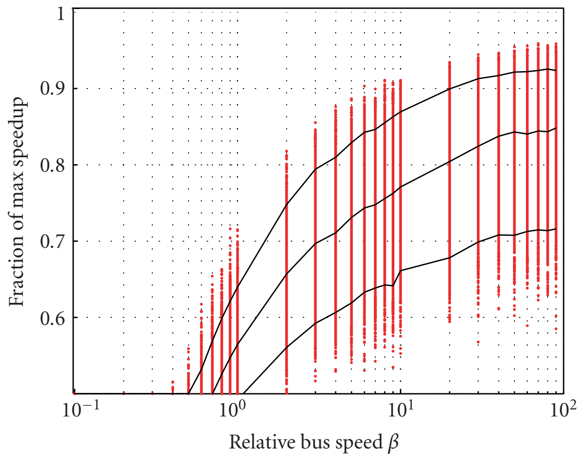

FIGURE 5: KL partitioning, no pipelining.

The KL algorithm has been introduced to Mod-SDR [30] as a remedy for this situation. Figure 5 shows its performance without pipelining. Astonishingly enough at a first glance, although the approach explicitly considers link cost (and it is even capable of trading load balance for link cost), the KL algorithm shows no systematic superiority compared to Hu's algorithm under these operating conditions. Speedup degradation in the low β range is a bit more graceful than in Figure 3, but at high bus speeds, the KL algorithm is easily outperformed by an approach as simple as Hu's. This can be explained by the fact that the partitioning approach by Kernighan and Lin indeed considers link cost, but tacitly assumes that bus transfers are nonconflicting at all times. Furthermore, its partitioning cut has an arbitrary orientation relative to the source node and the target node. As a consequence, a large number of LOGICAL_WAIT_IDLE conditions still occurs causing a large spread over the $[0.7; 0.9]$ range of speedup values.

As before, processor idle times associated with LOGICAL_WAIT_IDLE conditions can be completely eliminated by pipelining. Figure 6 shows the resulting performance of the KL algorithm. The contour lines reveal the highest speedup and the lowest speedup spread observed so far. Naturally, the speedup increases as the relative bus speed β increases, because link cost tends to be reduced and bus conflicts become less and less likely. Nevertheless, the results of this figure prove that there are better partitions than Hu's for all β. It can be concluded that the approach of Kernighan and Lin successfully effects a good compromise between maximum load balance and minimum link cost.

Since the KL algorithm is based on a local search of the design space (taking Hu's solution as a starting configuration), it may terminate in local optimum points. Global search methods, in contrast, should be able to avoid local optima and finally produce a better overall speedup. Spectral partitioning is a global search method, because it assesses the properties of the graph as a whole by operating on the matrix $\mathbf{W}$ of node weights and edge weights [33], and it is based on eigenvector computation for minimizing the cost of the partitioning cut [39]. What's more, $\mathbf{W}$'s diagonal elements $w_{m,m} = p_m$ are the nodes' processing runtimes according to (1) and its off-diagonal elements $w_{m,n} = 2 \cdot p_{\langle m,n \rangle}$ amount to the potential runtimes of bus transfer node pairs, where $p_{\langle m,n \rangle}$ is from (2). The weight matrix $\mathbf{W}$ is real-valued and symmetric, and spectral partitioning deliberately exploits this property [40, 41].

Figure 7 shows the performance results for spectral partitioning and nonpipelined scheduling. Unfortunately, these results are much worse than those of Kernighan and Lin; the 95% contour line just achieves 0.8 at high bus speeds, and the speedup spread remains large in the $[0.5; 0.8]$ interval. Clearly, such a behavior is completely unacceptable in practice; a fractional value of 0.5 means that the SDR implementation on the two-processor system meets real-time deadlines in the exact same way as on a single-processor system. Consequently, because the investment into the second processor does not pay off at all in the form of speedup, it must be concluded that the two-processor system is either ill-designed in its hardware or ill-conditioned in its operations.

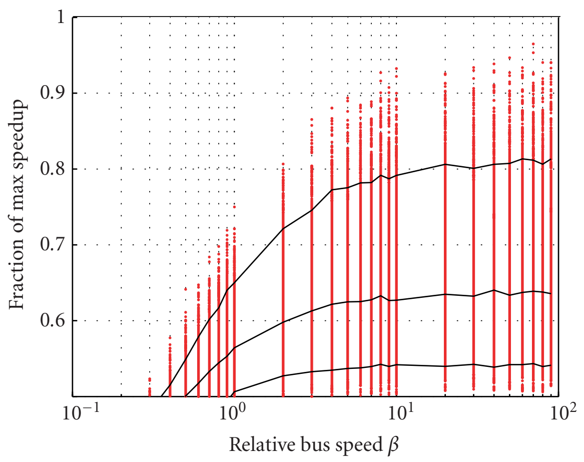

FIGURE 7: Spectral partitioning, no pipelining.

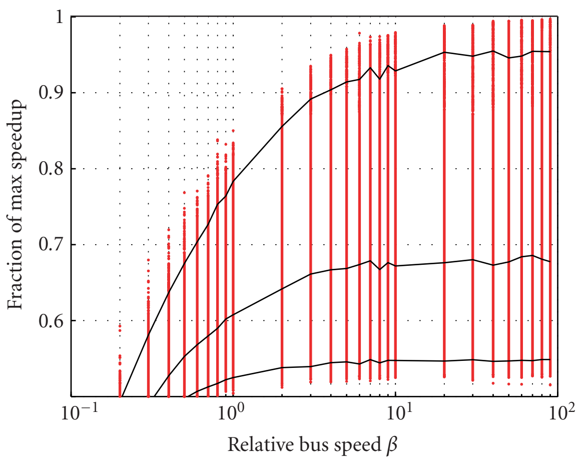

FIGURE 8: Spectral partitioning, pipelining.

Previous figures have shown that, as a matter of fact, better contour lines and narrower spread are possible using the same hardware and nonpipelined scheduling. Therefore, the reason for the inferior speedup behavior of spectral partitioning needs to be identified. Figure 8 shows its speedup results under partitioning. Obviously, only a small part of all realizations experience an improvement in speedup due to the elimination of LOGICAL_WAIT_IDLE conditions. Notably, the 5% contour line remains at the same speedup level for high bus speeds. These results back previous findings on spectral partitioning; the approach in its current form [33] does not generate well-balanced partitions. Load imbalance, as mentioned before in the context of Hu's algorithm, is a genuine feature of partitioning, not of scheduling. Failure to generate load-balanced partitions cannot be compensated for by any scheduling technique.

To sum up, the KL algorithm is able to outperform Hu's algorithm (but not systematically) in the low-to-midrange β region ($\beta < 5$), if frame-by-frame signal processing is the desired way of operating the software-defined physical layer of a Mod-SDR. True systematic superiority of the KL algorithm in speedup can only be observed under pipelining. A big disadvantage, however, is the depth of the pipeline; it depends on the graph structure, its actual value is not predictable, and it is quite a large integer number. (Histograms not shown graphically here: mean value around 20 at $M = 40$ nodes per graph, spread over a window of $[5; 35]$, independent of β for implicit Hu and spectral partitioning, depending on β for KL.) Memory demand increases linearly with the depth of the pipeline, and therefore pipelined operation in combination with the above approaches does not lead to workable Mod-SDR implementations.

In retrospect, frame-by-frame signal processing must be considered inadequate for circuit-switched services. There is simply no need to restrict partitioning algorithms to operating on a single graph copy anyway under these conditions. If the restriction is dropped, the QoS manager can approach partitioning and scheduling in a different, much simpler way. First of all, it turns out [35] that GDP (cf. Section 3) is optimal regarding delay; the depth of the pipeline is exactly 2 (or 3, if continuous RF transmission is to be automatically supported by the physical layer processing subsystem [38]). Second, GDP is optimal regarding speedup; it reaches the speedup limit $\bar{s}$, or the fractional value of 1.0 in Figures 4, 6, and 8, independent of β. In view of the previously discussed difficulties in *approaching* the limit, GDP is certainly the best design choice for circuit-switched services. If GDP's memory demand is still an issue, the QoS manager could easily resort to HFP. Its speedup performance for circuit-switched services (see [38, Figure 2]) is suboptimal, but totally comparable to that of the KL algorithm under pipelining (see Figure 6), however, at a constant pipeline depth of 2 and at less program and data memory demand than GDP's.

4.2. Packet-switched services

As mentioned in the beginning of Section 3, packet-oriented networks may require the QoS manager to operate on a single copy of the graph. Then the graph contains the complete set of computational tasks necessary for processing a single packet, but subsets of these tasks may be repetitive in nature. Taking the IEEE 802.11a wireless LAN standard as an example, it is easy to identify such tasks, even when looking at a single packet only: intercarrier/intraconstellation interleaving, constellation mapping, and IFFT [42]. All of these need to be repeated for every single OFDM symbol alike, just operating on different data within the packet. In contrast, nonrepetitive computations (per single packet) include scrambling and channel coding of IEEE 802.11a.

The speedup which could be expected under the conditions of the random graph model and a completely nonrepetitive task graph would be identical to that of Figures 3, 5, and 7. However, if a dominant subset of tasks were in fact repetitive, then pipelining approaches such as HFP and GDP could result in better speedup, when applied to the subset.

The following results of HFP and GDP for packet processing are conditional on the assumption that there are exactly N_F frames per packet which need to be processed identically. Furthermore, both processors are considered to be exclusively reserved for physical layer signal processing as soon as a packet has arrived. That is to say, one processor

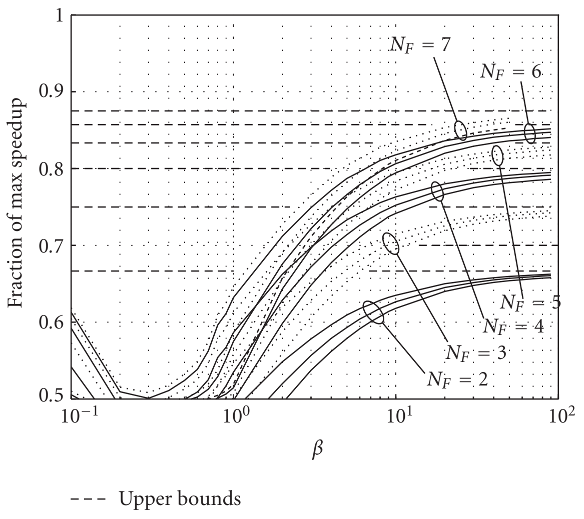

FIGURE 9: Half-frame pipelining, packet processing. Sample size 2000 per (β, N_F).

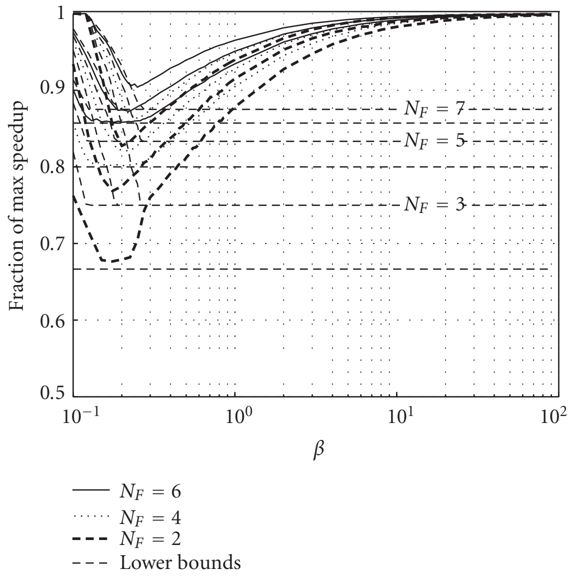

FIGURE 10: Graph duplication pipelining, packet processing. Sample size 2000 per (β, N_F).

cannot finish the remainder of some higher-layer computational task, while the other processor already starts processing the radio signals of a physical layer packet.

Figures 9 and 10 (adopted from [38]) show the speedup performance of HFP and GDP, respectively. For reasons of legibility, only the contour line triplets are drawn, parameterized by the number of frames per packet $N_F : 2 \leq N_F \leq 7$. In comparison to circuit-switched processing, GDP is no longer optimal, since the filling and the emptying of the pipeline cause idle times on the processors. A detailed discussion of HFP and GDP can be found in [38]. However, the crucial point in the above figures lies in the dashed lines representing upper bounds on HFP speedup, but lower bounds on GDP speedup. Therefore, it can be concluded that GDP systematically outperforms HFP in packet processing.

Even for reasonable values of β and low numbers of frames per packet (or task repetitions in parts of a graph), GDP closely approaches the speedup limit $\bar{s}$. So far, only transmissions with a constant N_F have been examined. However, IP traffic in real WLANs consists of a mix of packet sizes, and hence physical layer packets contain different numbers of radio frames. To gain more insight into this matter, packet size statistics of some tangible system have to be known. For the example of IEEE 802.11a, it has been shown [37] that small N_F (values of 10 and below) occur in the great majority of packets, so smart signal processing of small-sized packets is indeed an important issue in established WLAN standards. Given its superior speedup performance, GDP should be the first choice for packet-oriented signal processing.

5. SUMMARY AND CONCLUSION

As a starting point, some technical and commercial boundary conditions of SDR have been briefly reviewed. It follows from this account that certain design issues, which are related to real-time multiprocessing and modularity in flexible signal processing software, have been neglected at large in the existing SDR literature. Modular software-defined radio addresses these issues by looking into the organizational principles of signal processing rather than into the signal processing itself. Therefore, a novel way of modeling SDR software had to be introduced. Several techniques for mapping such software to hardware have been briefly reviewed. Mod-SDR system simulations presented in Section 4 allow to draw the following conclusions.

(i) With respect to *circuit-switched services*, frame-by-frame signal processing has proven inadequate. Pipelining methods such as GDP and HFP are to be preferred a priori. GDP is optimal regarding speedup and delay. HFP is suboptimal, but requires less memory than GDP. Therefore, HFP can only prove competitive against GDP if memory is a serious design issue. HFP can merely establish a compromise between the achievable speedup and dynamic power dissipation of the bus.

(ii) With respect to *packet-switched services*, frame-by-frame signal processing generally retains its right to exist. Pipelining is a viable alternative only if repetitive signal processing tasks can be identified. If so, GDP should be used. As for the repetitive task in isolation, HFP is systematically outperformed by GDP. Even frame-by-frame signal processing (which is independent of the number of repetitions) may show higher speedup than HFP.

Here, pipelining has been employed as a technique for software execution. However, additional work on Mod-SDR [34] provides strong hints that hardware subsystem pipelining also helps reducing dynamic power dissipation in CMOS hardware, at the same time keeping speedup high. Therefore, whenever signal processing in wireless communications is repetitive in nature, the insertion of pipelining is the preferable design guideline for Mod-SDR systems.

Future research directions include the improvement of spectral partitioning for direct comparison with the (non-pipelined) KL approach and a more comprehensive study of terminal behavior in packet-oriented networks. Further on, a suitable extension of the current hardware model to heterogeneous multiprocessor systems and interconnect topologies other than a bus would advance the design theory of modular software-defined radio.

REFERENCES

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

[2] *IEEE J. Select. Areas Commun.*, vol. 17, no. 4, 1999, Special Issue on Software Radio.

[3] *IEEE Commun. Mag.*, vol. 37, no. 2, 1999, Special Issue on Software Radio.

[4] A. Ivers and D. Smith, "A practical approach to the implementation of multiple radio configurations utilizing reconfigurable hardware and software building blocks," in *Proc. IEEE Military Communications Conference (MILCOM '97)*, vol. 3, pp. 1327–1332, IEEE, Monterey, Calif, USA, November 1997.

[5] A. Kountouris, C. Moy, L. Rambaud, and P. Le Corre, "A reconfigurable radio case study: a software based multi-standard transceiver for UMTS, GSM, EDGE and Bluetooth," in *Proc. IEEE Vehicular Technology Conference (VTC '01)*, vol. 2, pp. 1196–1200, Atlantic City, NJ, USA, October 2001.

[6] O. Faust, B. Sputh, D. Nathan, S. Rezgui, A. Weisensee, and A. Allen, "A single-chip supervised partial self-reconfigurable architecture for software defined radio," in *Proc. 17th International Symposium on Parallel and Distributed Processing (IPDPS '03)*, pp. 191–191, IEEE, Nice, France, April 2003.

[7] H. Miranda, P. Pinto, and S. Silva, "A self-reconfigurable receiver architecture for software radio systems," in *Proc. IEEE Radio and Wireless Conference (RAWCON '03)*, pp. 241–244, IEEE, Boston, Mass, USA, August 2003.

[8] A. Pacifici, C. Vendetti, F. Frescura, and S. Cacopardi, "A reconfigurable channel codec coprocessor for software radio multimedia applications," in *Proc. International Symposium on Circuits and Systems (ISCAS '03)*, vol. 2, pp. II-41–II-44, IEEE, Bangkok, Thailand, May 2003.

[9] T. Hentschel and G. Fettweis, "Sample rate conversion for software radio," *IEEE Commun. Mag.*, vol. 38, no. 8, pp. 142–150, 2000.

[10] W. Abu-Al-Saud and G. Stuber, "Efficient sample rate conversion for software radio systems," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM '02)*, vol. 1, pp. 559–563, IEEE, Taipeh, Taiwan, Republic of China, November 2002.

[11] W. Abu-Al-Saud and G. Stuber, "Modified CIC filter for sample rate conversion in software radio systems," *IEEE Signal Processing Lett.*, vol. 10, no. 5, pp. 152–154, 2003.

[12] J. Ming, H. Y. Weng, and S. Bai, "An efficient IF architecture for dual-mode GSM/W-CDMA receiver of a software radio," in *Proc. IEEE International Workshop on Mobile Multimedia Communications (MoMuC '99)*, pp. 21–24, IEEE, San Diego, Calif, USA, November 1999.

[13] J. Dodley, R. Erving, and C. Rice, "In-building software radio architecture, design and analysis," in *Proc. IEEE 11th International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC '00)*, vol. 1, pp. 479–483, IEEE, London, UK, September 2000.

[14] W. Schacherbauer, A. Springer, T. Ostertag, C. Ruppel, and R. Weigel, "A flexible multiband frontend for software radios using high IF and active interference cancellation," in *Proc. IEEE MTT-S International Microwave Symposium Digest (IMS '01)*, vol. 2, pp. 1085–1088, IEEE, Phoenix, Ariz, USA, May 2001.

[15] A. Wiesler, *Parametergesteuertes Software Radio für Mobilfunksysteme*, Ph.D. dissertation, Forschungsberichte aus dem Institut für Nachrichtentechnik, Universität Karlsruhe (TH), Karlsruhe, Germany, May 2001.

[16] M. Beach, J. MacLeod, and P. Warr, "Radio frequency translation for software defined radios," in *Software Defined Radio: Enabling Technologies*, W. Tuttlebee, Ed., pp. 25–78, John Wiley & Sons, London, UK, 2002.

[17] P. B. Kennington and L. Astier, "Power consumption of A/D converters for software radio applications," *IEEE Trans. Veh. Technol.*, vol. 49, no. 2, pp. 643–650, 2000.

[18] J. Singh, "High speed analog-to-digital converter for software radio applications," in *Proc. IEEE 11th International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC '00)*, vol. 1, pp. 39–42, IEEE, London, UK, September 2000.

[19] G. Ahlquist, M. Rice, and B. Nelson, "Error control coding in software radios: an FPGA approach," *IEEE Personal Communications*, vol. 6, no. 4, pp. 35–39, 1999.

[20] M. Valenti, "An efficient software radio implementation of the UMTS turbo codec," in *Proc. IEEE 12th International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC '01)*, vol. 2, pp. G108–G113, IEEE, San Diego, Calif, USA, September 2001.

[21] V. Thara and M. Siddiqi, "Power efficiency of software radio based turbo codec," in *Proc. IEEE Region 10 Conference on Computers, Communications, Control, and Power Engineering (TENCON '02)*, vol. 2, pp. 1060–1063, IEEE, Beijing, China, October 2002.

[22] A. Wiesler and F. K. Jondral, "A software radio for 2nd and 3rd generation systems," *IEEE Trans. Veh. Technol.*, vol. 51, no. 4, pp. 738–748, 2002.

[23] F. K. Jondral, "Parametrization—a technique for SDR implementation," in *Software Defined Radio: Enabling Technologies*, W. Tuttlebee, Ed., pp. 232–256, John Wiley & Sons, London, UK, 2002.

[24] J. Mitola, "Software radio architecture: a mathematical perspective," *IEEE J. Select. Areas Commun.*, vol. 17, no. 4, pp. 514–538, 1999.

[25] J. Mitola, "Software radios—survey, critical evaluation and future directions," in *Proc. National Telesystems Conference (NTC '92)*, pp. 13/15–13/23, IEEE, Washington, DC, USA, May 1992.

[26] C. Dick, "Reinventing the signal processor," *Xcell Journal*, vol. 45, pp. 72–75, Spring 2003.

[27] P. Galicki, "FPGAs have the multiprocessing I/O infrastructure to meet 3G base station design goals," *Xcell Journal*, vol. 45, pp. 80–84, Spring 2003.

[28] "Software communications architecture specification, jtrs-5000sca v2.2.1," Joint Tactical Radio System (JTRS) Joint Program Office, April 2004, [Online] available: http://jtrs.army.mil.

[29] A.-R. Rhiemeier and F. K. Jondral, "Mathematical modeling of the software radio design problem," *IEICE Transactions on Communications*, vol. E86-B, no. 12, pp. 3456–3467, 2003, Special Issue on Software Defined Radio Technology and Its Applications.

[30] A.-R. Rhiemeier and F. K. Jondral, "A software partitioning algorithm for modular software defined radio," in *Proc. 6th International Symposium on Wireless Personal Multimedia Communications (WPMC '03)*, pp. 42–46, Yokosuka, Japan, October 2003.

[31] A.-R. Rhiemeier and F. K. Jondral, "On the design of modular software defined radio systems," in *Proc. IEE Colloquium*

on DSP Enabled Radio, Institute for System Level Integration (ISLI), Alba Campus, Livingston, Scotland, UK, September 2003.

[32] A.-R. Rhiemeier and F. K. Jondral, "Enhanced resource utilization in software defined radio terminals," in *Internationales Wissenschaftliches Kolloquium (IWK '03)*, Technische Universität, Ilmenau, Germany, September 2003.

[33] U. Berthold, A.-R. Rhiemeier, and F. K. Jondral, "Spectral partitioning for modular software defined radio," in *IEEE 59th Vehicular Technology Conference (VTC '04)*, vol. 2, pp. 1218–1222, Milano, Italy, May 2004.

[34] A.-R. Rhiemeier and F. K. Jondral, "Software partitioning and hardware architecture for modular SDR systems," in *Proc. Software Defined Radio Technical Conference and Product Exposition (SDR '03)*, vol. 2, pp. 9–15, SDR Forum, Orlando, Fla, USA, November 2003.

[35] U. Berthold, A.-R. Rhiemeier, and F. K. Jondral, "A pipelining approach to operating modular software defined radio," in *Proc. IEEE/Sarnoff Symposium on Advances in Wired and Wireless Communication (SARNOFF '04)*, pp. 201–204, Princeton, NJ, USA, April 2004.

[36] A.-R. Rhiemeier, "A comparison of scheduling approaches in modular software defined radio," in *Proc. 3rd Karlsruhe Workshop on Software Radios (WSR '04)*, pp. 33–38, Karlsruhe, Germany, March 2004, also appeared as reprint in: *Frequenz: Journal of Telecommunications*, vol. 58, no. 5/6, pp. 115–120, 2004.

[37] A.-R. Rhiemeier, T. Weiss, and F. K. Jondral, "Half-frame pipelining for modular software defined radio," in *Proc. IEEE 15th International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC '04)*, vol. 3, pp. 1664–1668, IEEE, Barcelona, Spain, September 2004.

[38] A.-R. Rhiemeier, T. Weiss, and F. K. Jondral, "A simple and efficient solution to half-frame pipelining for modular software defined radio," in *Proc. Software Defined Radio Technical Conference and Product Exposition (SDR '04)*, vol. A, pp. 119–125, SDR Forum, Phoenix, Ariz, USA, November 2004.

[39] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon, "A min-max cut algorithm for graph partitioning and data clustering," in *Proc. IEEE International Conference on Data Mining (ICDM '01)*, pp. 107–114, San Jose, Calif, USA, November 2001.

[40] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak Mathematical Journal*, vol. 23, no. 98, pp. 298–305, 1973.

[41] M. Fiedler, "A property of eigenvectors of non-negative symmetric matrices and its application to graph theory," *Czechoslovak Mathematical Journal*, vol. 25, no. 100, pp. 619–633, 1975.

[42] "ANSI/IEEE Std 802.11, Wireless LAN MAC and PHY specifications," 1999 Edition, and IEEE 802.11a-1999, High-speed Physical Layer in the 5 GHz Band.

Arnd-Ragnar Rhiemeier received a first degree in electrical engineering from the Ruhr-Universität Bochum, Germany, in 1995, then continued his studies at the Universität Karlsruhe (TH), Germany. Supported by a grant from the German Academic Exchange Service (DAAD), he spent two terms at the National Institute of Applied Sciences, Lyon, France, in 1996 and 1997, working in the field of pattern recognition. In 1998, he resumed his graduate studies at the Institut für Nachrichtentechnik, Universität Karlsruhe (TH). In 1999, he completed his final project at the Center for Communications and Signal Processing Research (CCSPR), New Jersey Institute of Technology, Newark, NJ, USA, and received a Dipl.-Ing. degree in electrical engineering from the Universität Karlsruhe (TH), Germany. Subsequently, he committed himself to a teaching and research assistantship position at the Institut für Nachrichtentechnik. In 2004, he received a Ph.D. degree summa cum laude in telecommunications for his work on software-defined radio architectures and algorithms. His current professional interests include design flow methodologies and the productization of advanced concepts in communications such as MIMO and software-defined radio. He has been an IEEE Member for 11 years and served as the Chairman of the IEEE Student Branch Karlsruhe for two years.