To appear in the Journal of Signal Processing Systems.

PARAMETERIZED SETS OF DATAFLOW MODES AND THEIR APPLICATION TO IMPLEMENTATION OF COGNITIVE RADIO SYSTEMS

Shuoxin Lin, Lai-Huei Wang, Aida Vosoughi, Joseph R. Cavallaro, Markku Juntti, Jani Boutellier, Olli Silvén, Mikko Valkama, Shuvra S. Bhattacharyya

S. Lin
Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742, USA.
E-mail: slin07@umd.edu

L. Wang
Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742, USA.
E-mail: laihuei@umd.edu

A. Vosoughi
Department of Electrical and Computer Engineering,
Rice University, Houston, TX 77005, USA
E-mail: vosoughi@rice.edu

J. Cavallaro
Department of Electrical and Computer Engineering,
Rice University, Houston, TX 77005, USA
E-mail: cavallar@rice.edu

M. Juntti
Department of Communications Engineering, and
Department of Computer Science and Engineering,
University of Oulu, Finland FI-90014
E-mail: markku.juntti@ee.oulu.fi

J. Boutellier
Department of Communications Engineering, and
Department of Computer Science and Engineering,
University of Oulu, Finland FI-90014
E-mail: jani.boutellier@ee.oulu.fi

O. Silvén
Department of Communications Engineering, and
Department of Computer Science and Engineering,
University of Oulu, Finland FI-90014
E-mail: olli.silven@ee.oulu.fi

M. Valkama
Department of Communications Engineering,
Tampere University of Technology,
Tampere, Finland FI-33101

Abstract Cognitive radio networks present challenges at many levels of design, including configuration, control, and cross-layer optimization. To meet requirements of bandwidth, flexibility and reconfigurability, systematic methods to model and analyze cognitive radio designs on signal processing platforms are desired. To help address these challenges, we present in this paper a novel dataflow modeling technique, called *parameterized set of modes* (*PSM*). PSMs allow efficient representation, manipulation and application of related groups of processing configurations for functional design components in signal processing systems. PSMs lead to more concise formulations of actor behavior, and a unified modeling methodology for applying a variety of techniques for efficient implementation. We develop the formal foundations of PSM-based modeling, and demonstrate its utility through two case studies involving the mapping of reconfigurable wireless communication functionality into efficient implementations.

Keywords Cognitive radio · dataflow graphs · embedded signal processing · heterogeneous multiprocessors · model-based design.

1 Introduction

Cognitive Radio is an emerging technology that enables a wireless transceiver to cognitively manage its wireless spectrum for improved agility and efficiency. Flexibility and reconfigurability of the implementation at various layers, including RF, baseband, and MAC layers, with cross-layer modeling and control, will be important to realize the efficiency potential of spectrum sharing. Realizing the potential of cognitive radio will also require transceivers to dynamically reconfigure communication parameters based on multidimensional criteria, including channel conditions, link performance, and user requirements. Meanwhile, increasing bandwidths and data rates pose new challenges to the baseband (BB) processing chain, as well as to radio frequency (RF) processing.

The above challenges motivate important new research directions in both software- and hardware-based design of wireless communication systems. On the software side, software defined radio (SDR) now utilizes various kinds of high-performance computing devices, ranging from multi-core programmable digital signal processors (PDSPs), streaming SIMD extensions (SSE), to general purpose graphics processing units (GPGPUs). On the hardware side, programmable baseband computation and related design chains have significantly developed to enable more efficient control of computational resources and hardware. However, practical and systematic approaches to reconfiguration based on programmable paradigms are still lacking. For example, software-based adaptive configuration of radio frequency chains is still in its infancy, but is a key ingredient of the frequency agile radios needed for cognitive devices and flexible RF spectrum use. The trend of increasing diversity and flexibility in both the functionality and the

E-mail: mikko.e.valkama@tut.fi

S. S. Bhattacharyya
Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies,
University of Maryland, College Park, MD 20742, USA.
E-mail: ssb@umd.edu

computational platforms of wireless systems results in complex design spaces that must be considered during design and implementation. The complexity of these design spaces and their novel constraints strongly motivate the development of new design methodologies.

Dataflow models offer a promising foundation for such design methodologies in part because they provide scalable and retargetable representations of signal processing applications [3]. Designers can migrate a common dataflow model of an application across different types of computing platforms, while changes are localized to the implementations of individual actors (dataflow-based functional components). The scalability and retargetability of dataflow models reduces the designer's effort in debugging, validating, and fine-tuning a complex signal processing application that must satisfy stringent, multi-dimensional constraints.

To express dynamics in complex signal processing applications, a number of dynamic dataflow models have been proposed, including *parameterized synchronous dataflow* (*PSDF*) [2], *Boolean dataflow* (*BDF*) [7], and *core functional dataflow* (*CFDF*) [15]. PSDF provides semantics to manipulate application parameters in dataflow models at run-time. BDF introduces special control actors to allow data-dependent invocation of actors. CFDF applies the concept of actor "modes", where different modes can have differing dataflow behavior, and mode transitions can be data-dependent. CFDF is tailored to natural design of actors with dynamic functionality, and facilitates prototyping of dataflow applications, as well as identification of more specialized dataflow behaviors [16], such as BDF, cyclo-static dataflow (CSDF) [5] or synchronous dataflow (SDF) [13].

When using CFDF, a designer specifies the behavior of the different modes of each CFDF actor, and the transitions among these modes. However, as the number of modes grows and the mode transitions become more complex, CFDF formulations can become unwieldy in terms of actor specification, analysis and implementation. In this paper, we present a novel modeling method, called *parameterized set of modes* (*PSM*), which is a high-level abstraction that efficiently represents parameterized functionality within groups of related modes for CFDF actors. PSMs enable novel ways for representing, manipulating and applying related
groups of actor modes that lead to more concise formulations of actor behavior, and a unified modeling methodology for applying a variety of techniques for efficient implementation. We develop the formal foundations of PSM-based modeling, and demonstrate its utility through two case studies involving the mapping of reconfigurable wireless communication functionality into efficient implementations.

2 Background and Related Work

2.1 Background

Dataflow modeling has proven to be valuable in allowing designers of signal processing systems to describe applications in an intuitive and structured manner [3]. As system complexity increases, coarse-grained, dynamic dataflow models have gained increasing significance for their flexibility and their power in exposing high level application structure that is relevant for deriving optimized implementations.

Core functional dataflow (CFDF) is a deterministic sub-class of enable-invoke dataflow (EIDF) [15] in which dynamic functionality in an actor is specified as a set of actor *modes*. In each mode, the actor possesses deterministic dataflow behavior, meaning that the production/consumption rates on all actor output/input ports are known, constant values. Upon each invocation (actor firing), the actor executes in its current mode, and in addition to consuming input tokens and producing output tokens, the actor selects one *next mode* from its set of modes. This next mode determines the mode in which the next actor invocation executes (unless the actor mode is reset or otherwise overridden by the controlling scheduler). The next mode determined during an actor invocation can be fixed (known at compile time) or data dependent.

As the level of dynamic behavior in each actor grows, the size of the actor's mode set may increase significantly. For an actor with a large number of parameters and corresponding variations in functionality, a large set of modes can be difficult to specify, analyze, and map into efficient implementations. Such parameterized modes can arise when applying CFDF to model cognitive radio applications, for example, due to the handling of different communication modes, sample rates, or filter configurations. In this paper, we address the problem of efficient integration of parameterization into the mode-based structure of CFDF actor models.

2.2 Related Work

A technique called *mode grouping* for CFDF graphs has been developed in [14]. It is demonstrated that mode grouping can improve scheduling results by aiding the discovery of statically schedulable subgraphs. In [18], CFDF is applied in simulation of dynamic communication systems. CFDF modeling is also applied as the semantic basis for the lightweight dataflow design environment, which is introduced and applied to design and implementation of wireless communication systems in [19]. These works apply the CFDF model in various useful ways, but are unable to streamline their associated analysis or implementation when manipulating groups of modes that are related through parameterization. The mode-based parameterization techniques introduced in this paper are developed to bridge this gap.

Various research efforts have been directed towards integrating dynamic behavior into dataflow models.
In [12], a design framework called SysteMoc is developed for applying dataflow structures, similar to those used in CFDF, involving guarded invocations and state transitions specified by finite state machines (FSMs). The work also includes design space exploration and code synthesis for FPGA platforms. In [21], a dataflow based analysis method is proposed for SDR applications. This method adopts the concept of "SDF scenarios" to incorporate some degree of dynamism for better estimation of system resource requirements and throughput. Moreover, methods for quasi-static scheduling of statically-schedulable sub-graphs within larger dynamic dataflow graphs are explored in [10].

In the context of the related work described above, the main contributions of this paper are described as follows. We enhance the CFDF model of computation by introducing the concept of parameterized set of modes (PSM), which incorporates dynamic parameterization into actor modes, thereby increasing the effec-

tiveness with which designers can design and implement CFDF-based, dynamic dataflow models for signal processing systems. PSM-based modeling of actors provides a common framework for integrated specification, analysis and implementation that deeply integrates mode- and
parameter-based actor characterizations. Although we develop the PSM model in the context of CFDF in this paper, we envision that the ideas can be adapted to related dataflow modeling and programming techniques, such as, for example, SysteMoc [12] and CAL [9]. Exploring and applying such adaptations is a useful direction for future work.

3 Parameterized Set of Modes

In this section, we define a new modeling concept, called *parameterized set of modes* (*PSM*), which is motivated in Section 2 as a method for incorporating dynamically parameterized modes efficiently into the CFDF modeling framework.

3.1 Notation

To develop the PSM concept precisely, we first introduce some notation and review the definition of the CFDF model of computation. For a given dataflow graph actor A, we denote the set of input ports of A by $in(A)$. We also denote the set of nonnegative integers by N, and the set of Boolean values by B. We denote the values in B as *true* and *false*.

When using PSMs, we allow CFDF actors to have arbitrary sets of parameters. Following notation similar to that of parameterized dataflow graphs [2], we denote the set of parameters of a given actor A as $param(A)$, and for each parameter in $p \in param(A)$, we denote the set of permissible values of p as $domain(p)$. At any given point during dataflow graph execution, an actor parameter p has associated with it a unique parameter value $v \in domain(p)$, which is referred to as the *configuration* of p at that point in time. A *configuration* for A can then be specified as a set of configurations for all of the parameters in $param(A)$. Some combinations of possible parameter values may be considered invalid because they do not make sense together. The set of all valid configurations for A is denoted as $DOMAIN(A)$. At a given point during execution, the specific configuration for A that is determined by its current parameter values is referred to as the *active configuration* of A. Similarly, the specific mode that a CFDF actor is in during a given firing is referred to as the *active mode* for the actor.

If S_1 and S_2 are sets, then by $S_1 \subset S_2$, we mean that S_1 is a subset (not necessarily a proper subset) of S_2. Thus, S_1 can be empty, equal to S_2, or a proper subset of S_2.

3.2 Review of CFDF Semantics

As introduced in [15], each CFDF actor A is characterized by a nonempty set M_A of modes in which it can execute, and for any given mode $m \in M_A$, the actor A consumes a fixed number of tokens per firing on each input port, and produces

a fixed number of tokens per firing on each output port. These production and consumption rates may vary across different modes, but must be constant for any given mode. Each CFDF actor A is also characterized by its *enabling function* ε_A, which determines whether or not, based on a given set of token populations on its input FIFOs, A is enabled. If A has at least one input port, then this enabling function can be viewed as a mapping

$$\varepsilon_A : (T_A \times M_A) \rightarrow B, \tag{1}$$

where $T_A = N^{|in(A)|}$ denotes the set of all possible buffer populations for input ports of A (assuming some underlying ordering of these ports) [15]. If A has no input ports, then its enabling function reduces simply to the Boolean constant *true*. The CFDF formulation of enabling functions can easily be generalized to take into account finite-capacity output buffers (i.e., by requiring sufficient free space on output buffers before allowing an actor to be fireable). For brevity and clarity, we suppress these details of bounded buffer CFDF execution in this paper, and we simply assume that FIFOs have unbounded token capacity, unless otherwise stated.

3.3 Motivation for Parameterized Sets of Modes

The CFDF formulation can become unwieldy when working with parameterized actors that have large parameter sets, especially if one or more actor parameters can affect the production and consumption rates of an actor. For example, consider a parameterized downsampler actor that provides an $N : 1$ downsampling of its input signal. Such an actor requires N distinct modes in its CFDF specification even though the operation of all N alternative modes have closely related (parameterized) functionality. Using the PSM concept introduced in this section, we can group all of these related modes together into a single *mode set* σ, where the individual mode in σ that is active during any given actor firing is determined uniquely by the actor parameter set (in this case, by the parameter N).

As a slightly more elaborate example, consider an actor S that can function either as a downsampler or an upsampler depending on its configuration. Such an actor could be useful, for example, as part of a programmable, multistage subsystem for sample rate conversion. This actor can be parameterized with two parameters u and N, where u is Boolean-valued and indicates whether or not S functions as an upsampler (if $u = \mathit{false}$, then the actor functions as a downsampler), and N provides the upsampling or downsampling factor. Using the PSM concept, this actor can be specified precisely using two mode sets — one for the upsampling-related modes, and the other for the downsampling-related modes. In any given mode set, the production and consumption rates are determined uniquely by the actor parameters. For example, in the mode set associated with upsampling ($u = \mathit{true}$), $N = 3$ yields a consumption rate of 1 and production rate of 3.

Intuitively, a *PSM-enhanced CFDF* specification, or *PSM-CFDF* specification, allows an actor's modes to be grouped into "clusters" or sets that have related functionality, and are therefore efficient to work with as distinct units — e.g., in terms of design tasks such as specification, analysis, optimization, profiling, and

integration. In general, the actor groups may overlap, but collectively, they should "cover" the entire set of modes of the associated CFDF actor. Furthermore, the actor groups in a PSM-based specification should be related uniquely to the actor modes through the parameters of the given actor.

3.4 Formal Definition of PSM-CFDF

Given a PSM-CFDF actor A with mode set M_A, a PSM ρ for A is a 3-tuple $\rho = (S, C, f)$, where $S \subset M_A$, $C \subset DOMAIN(A)$, and $f : C \to S$. The set C, denoted as $psa\_domain(\rho)$, can be viewed as the set of possible actor configurations when the actor is firing in mode set S. The set S, denoted $psa\_modeset(\rho)$, is the set of modes in actor A that is associated with ρ — i.e., whenever A fires in PSM ρ, it fires one of the modes within $psa\_modeset(\rho)$. Finally, the mapping f, denoted F_ρ, specifies the unique mode within $psa\_modeset(\rho)$ that is active whenever A executes in mode set S and a given actor configuration is active.

Given a PSM-CFDF actor A with mode set M_A, and a set R of PSMs for A, we say that R *covers* A if every mode in M_A is contained in the mode set of at least one element of R — that is, if

$$M_A = \bigcup_{\rho \in R} psa\_modeset(\rho). \tag{2}$$

A PSM-CFDF actor A is a CFDF actor with an associated set R of PSMs that covers A, and a family of mappings $\{psa\_next_{r,c} : I(F_r(c)) \to R \mid r \in R \text{ and } c \in DOMAIN(A)\}$. Here, for a given mode $m \in M_A$, $I(m)$ denotes the set of all possible combinations of inputs — i.e., all possible n-tuples of token vectors, where $n = |in|$, and the size of (number of elements in) each token vector is equal to the consumption rate of the corresponding port in mode m.

In other words, for each pair (r, c), there is a mapping $psa\_next_{r,c}$, called the *next PSM function of PSM r under actor configuration c*, that determines uniquely a specific mode $m\prime$ for any given input data set for that mode; this mode $m\prime$ can be interpreted as the *next PSM* for the actor — i.e., the PSM that should be active for the next firing of A.

For a PSDF-CFDF actor A, we denote the associated set of PSMs at $PSMset(A)$, and the associated family of mappings as $mappings(A)$.

The next PSM function is related to the invoking function of A, as defined by CFDF semantics. In particular, for a given actor firing, the next mode, as determined by the invoking function, should agree with (be an element of) the next PSM, as determined by $psa\_next(r, p)$. For details on the CFDF invoking function, we refer the reader to [15].

The concept of PSM is a level of abstraction that helps the designer to better understand and expose connections between the actor's parameters and modes. PSM analysis can be combined with various processes in a design framework, such as scheduling and processor selection, to name a few. By grouping into a single PSM the modes of an actor that share some common property, a designer can manipulate the associated modes and apply aspects of the property in an integrated and systematic way.

3.5 PSM Transition Graph

For a PSM-CFDF actor, the next PSM function defines the range of modes in which the actor executes in its next invocation. The structure of transitions among PSMs therefore can provide valuable information about the actor's dynamic behavior. These transitions can be expressed formally by a construction that we call the *PSM transition graph*.

The PSM transition graph for a PSM-CFDF actor A is a directed graph $G_{psm} = (V_{psm}, E_{psm})$, where V_{psm} is the set of vertices and E_{psm} is the set of edges. The set of vertices is in one to one correspondence with the PSMs of A; the PSM transition graph vertex associated with PSM r is denoted as $v_{psm}(r)$. Two PSM transition graph vertices $v_{psm}(x)$ and $v_{psm}(y)$ are connected by a directed edge $e = (v_{psm}(x), v_{psm}(y))$ if there exist an input vector ν and a configuration c such that $y = psa\_next_{x,c}(\nu)$. Such an edge e is annotated with a *label*, $label(e) = c$. Note that multiple edges can have the same label if different next PSMs are "reachable" from the same current PSM and same configuration under different input vectors. Compared to finite state machine (FSM) representation of state transitions, the PSM transition graph contains higher level information on the structure of PSMs. Such higher level structure may be difficult to extract or intuitively understand from conventional FSM-style representations (i.e., where each mode corresponds to a separate FSM state), especially when the number of modes is large or their connections are irregular.

Figure 1(b) shows an example of a PSM transition graph. Further details about the actor in this example are discussed in Section 5.4.

3.6 Implementation Considerations

When implementing a PSM-CFDF actor, we do not anticipate that designers will typically need to explicitly implement the mappings (mathematical functions) F_ρ and $psa\_next\{r, p\}$. These mappings are useful as analytical tools, but their explicit realization in software is not in general essential for the PSM-CSDF model — e.g., an actor designer would not need to provide a software function/method or hardware description language module that is dedicated to implementing each of these mappings. Instead, for example, critical aspects of F_ρ may be validated through unit testing, and the next PSM may be determined as a by-product of actor firing — e.g., through an actor-level application programming interface (API) that is used by schedulers to invoke the actor.

3.7 Application Example

In this section, we show an example of applying PSM-CFDF concepts in actor design for a reconfigurable OFDM demodulator that is geared towards cognitive radio systems. Such systems can involve significant
amounts of parameterization in actor designs. Figure 1(a) shows a parameterized demodulator actor that supports different operational modes, including QPSK and QAM16. The actor maps the B samples into an $M \times B$ bit stream. This actor

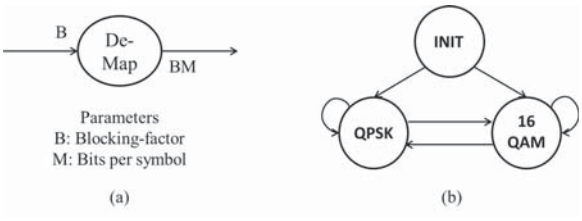

Fig. 1 An example of a PSM-CFDF actor: OFDM demapper example. (a) Actor interface. (b) PSM transition graph.

has two parameters: M for the number of bits per sample, and B for the vectorization degree (see [17] for fundamental developments on actor-level vectorization for signal processing dataflow graphs). Since M represents the number of bits for each symbol, $M = 2, 4$ correspond, for example, to QPSK, QAM16, respectively. B can take on any integer value between 1 and B_{max}, where B_{max} is the maximum vectorization degree (e.g., as a designer or design tool might set based on memory constraints). The parameter B allows symbols to be buffered and processed together in batches (block processing). For example, if $B = 1$, then each actor invocation processes a single input symbol; if $B = 10$, then 10 symbols are buffered and processed together in one invocation.

The de-mapper in Figure 1(a) is modeled as a PSM-CFDF actor A as follows. Actor configurations are specified in the form (M, B). The set of modes of the actor is given as:

$$M_A = \{INIT, QPSK_1, QPSK_2, \ldots, QPSK_{B_{max}}, \\ QAM16_1, QAM16_2, \ldots, QAM16_{B_{max}}\} \tag{3}$$

Based on the functionality, M_A can be clustered into 3 PSMs: $\{\rho_i = (S_i, C_i, f_i) \mid i = 1, 2, 3\}$, where $S_1 = \{QPSK_n \mid 1 \leq n \leq B_{max}\}$, $C_1 = \{(2, n) \mid 1 \leq n \leq B_{max}\}$, $f_1(M, B) = QPSK_B$; $S_2 = \{QAM16_n \mid 1 \leq n \leq B_{max}\}$, $C_2 = \{(4, n) \mid 1 \leq n \leq B_{max}\}$, $f_2(M, B) = QAM16_B$; $S_3 = \{INIT\}$, $C_3 = \{(m, n) \mid m = 2, 4; 1 \leq n \leq B_{max}\}$, $f_3(M, B) = INIT$.

Based on this decomposition into PSMs, Figure 1(b) illustrates the PSM transition graph for the demapper actor. Upon initialization or reset, the actor enters the *INIT* mode, the only mode in ρ_3. After initialization, the actor enters a mode in ρ_1, or ρ_2, based on the configuration. For any mode in ρ_1, the ratio of the production rate $prd(A)$ to the consumption rate $cons(A)$ is 2. Similarly, for any mode in ρ_2, $prd(A)/cons(A) = 4$. To avoid clutter in the diagram, edge labels are not shown.

3.8 Summary

In this section, we have presented an enhancement to the framework of CFDF modeling called parameterized set of modes (PSM), and we have introduced the PSM-CFDF approach to the modeling of dynamic dataflow actors with dynamically variable parameters. To illustrate the approach, we have presented a detailed

example of an OFDM demapper actor that is modeled in terms of PSM-CFDF semantics. This example and its associated PSM transition graph representation concretely illustrate the novel form of higher level modeling structure that is exposed by the PSM modeling concept and the associated PSM-CFDF design methodology.

4 PSM-level Static Scheduling for CFDF Graphs

In this section, we demonstrate the application of PSM to efficient scheduling of CFDF-based programs.

A general scheduling approach for CFDF graphs is the so-called *canonical scheduling* approach discussed in [16]. In canonical scheduling, a sequential ordering L of the dataflow graph actors is constructed [16]. At run-time, the scheduler iteratively traverses the list L, and upon visiting each actor A, the scheduler checks the enabling condition (availability of sufficient input data) for A, and invokes A if the enabling condition is satisfied. This scheduling approach is useful in the sense that it is very general (applicable to any CFDF graph), easy to understand, and easy to implement. However, the efficiency of canonical scheduling can be relatively low because of the frequency with which enabling conditions must be checked.

4.1 Statically Schedulable Regions

Static schedules, where the sequence of actor firings is deterministic and unconditional (not guarded by actor-level checking of enabling conditions) can be significantly more efficient and predictable compared to dynamic scheduling approaches, such as canonical scheduling. Even if the overall dataflow graph does not allow for static scheduling (due to the presence of dynamic dataflow), it may be possible to identify "statically schedulable regions" of the graph — i.e., parts of the graph that can be scheduled statically. Such regions can be scheduled using efficient static scheduling techniques, which have been developed extensively in the literature (e.g., see [3]), and then the static schedules for the different regions can be integrated through a "top-level" dynamic scheduling mechanism.

In this section, we develop PSM-based methods for constructing and applying statically schedulable regions for efficient implementation of CFDF graphs. The concept of statically schedulable regions itself is not new, and has been studied in depth, for example, in the implementation of CAL programs [11]. Our contribution in this section, which we refer to as *PSM-level static scheduling*, is to demonstrate methods for integrating the concepts of PSMs and statically schedulable regions, therefore combining the benefits of both approaches, and enabling structure exposed from PSMs to help guide the construction of efficient schedules. More specifically, in our development of PSM-level static-scheduling, we utilize information about actor parameters to form hierarchical PSMs, where each hierarchical PSM is constructed based on combinations of actor modes that share common scheduling properties.

In the remainder of this section, we outline our proposed PSM-level static scheduling approach and present experimental results on an application example.

4.2 PSM-level Static Scheduling

PSM-level static scheduling is a hierarchical scheduling technique, where subgraphs within a dataflow specification are combined into *hierarchical actors*, and execution of a hierarchical actor corresponds to execution of a schedule for the associated subgraph. If H is a hierarchical actor with associated subgraph G, we say that H *encompasses* G, and G is the *nested subgraph* of H.

In the class of CFDF-PSM specifications addressed in this work, a hierarchical actor contains a set of modes, and can also contain a set of PSMs, just as non-hierarchical (leaf-level) actors. In the case of a hierarchical actor H, each mode m of H corresponds, respectively, to a mapping $Z_m : V_e \rightarrow \gamma$, where $G_e = (V_e, E_e)$ denotes the graph encompassed by H, γ is the set of all actor modes across all actors in V_e, and $Z_m(v) \in M_v$ for all v. Recall here that M_v represents the set of modes for a given actor v.

Intuitively, execution of H in a given mode $m \in M_H$ corresponds to execution of the encompassed graph with all actors operating in the modes specified by Z_m. The duration (termination criterion) of such an execution is a design issue associated with the construction of H, similar in some ways to the concept of "subsystem iteration" in parameterized dataflow [2]. In this paper, we assume that each execution of H in a given mode m corresponds to execution of a minimal static periodic schedule of the SDF graph, denoted $G_{sdf}(H, m)$, that results from fixing the actors in G_e based on the mode assignments specified by Z_m. Exploration of other kinds of termination criteria in this context is a useful direction for further work.

In our development of PSM-level static scheduling in this paper, we assume that the hierarchical actors employed are provided as part of the specification — i.e., as part of the design hierarchy. Another interesting direction for future work is in the development of automated methods to group (cluster) subgraphs into hierarchical actors for PSM-level static scheduling.

4.3 Construction of SDF Scheduling PSMs

Building on the concepts introduced in Section 4.2, we introduce a simple method to partition the mode set M_H of a hierarchical actor H in a manner that facilitates construction of statically schedulable regions. This leads to a unique partitioning of M_H into a set of PSMs that we refer to as *SDF scheduling PSMs*. The method is useful in systematically decomposing the structure of a hierarchical PSM-CFDF actor in a manner that that captures subsystem-level, multi-mode behavior that is common in cognitive radio systems.

The process of constructing SDF scheduling PSMs operates by iterating through all modes in H, and dividing the modes into subsets (PSMs) $S_1, S_2, \ldots, S_k$, where all modes in a given S_i correspond to the same SDF repetitions vector for the encompassed graph $G(e)$. In other words, if $m_1, m_2 \in S_i$, and $a \in V_e$, then $q_1(a) = q_2(a)$, where q_1 and q_2 denote, respectively, the SDF repetitions vectors of $G_{sdf}(H, m_1)$ and $G_{sdf}(H, m_2)$. The resulting mode sets $S_1, S_2, \ldots, S_k$ are then parameterized with one more scheduling parameters that can be configured and adapted based on considerations such as the given performance constraints, repetitions vectors q_i, and structure of $G(e)$. This process depends on fundamental

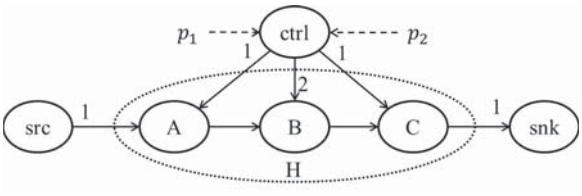

Fig. 2 A synthetic CFDF graph that is used to illustrate PSM-level static scheduling concepts.

properties of the SDF repetitions vector and requires that the set of SDF graphs $\{G_{sdf}(H, m) \mid m \in M_h\}$ satisfy SDF consistency conditions. For details on SDF fundamentals and consistency conditions, we refer the reader to [13].

In cognitive radio systems, actors can often be configured statically or dynamically by various parameters, resulting in large sets of possible actor modes. If the actors' mode spaces are viewed independently, the total number of possible mode combinations to consider can grow exponentially, making the system unwieldy and inefficient for scheduling analysis. The integration of PSM techniques to hierarchical CFDF modeling techniques, as introduced in this section, introduces an alternative, more compact designs space — the design space of scheduling parameters for the PSMs $S_1, S_2, \ldots, S_k$ — that facilitates efficient scheduling, including the application of SDF scheduling techniques to statically schedulable regions.

4.4 Synthetic Example

To illustrate the PSM-level static scheduling technique introduced in Section 4.2 and Section 4.3, Figure 2 shows a synthetic CFDF graph with 2 parameters, p_1 and p_2. Intuitively, the parameters p_1 and p_2 control (select) the modes of A and C, respectively, and p_1 and p_2 together control the mode of B. The parameter values and their corresponding actor modes, production rates, and consumption rates are shown in Table 1. Here, the special actor *ctrl* reads parameter values from an input source (e.g., a file), checks their validity, and sends them as tokens to A, B and C.

Now suppose that H is a hierarchical actor that encompasses the subgraph associated with actors A, B, and C. The actors enter "initialization modes" A_0, B_0 and C_0, respectively, upon system reset, and wait for parameter tokens that are passed from *ctrl*. After receiving the parameter values, the actors continue to their respective operational modes, as specified by the received parameters, until all data from *src* has been processed.

Analyzing the repetitions vectors in M_H, and the mode space of H, and constructing SDF scheduling PSMs leads to the PSMs outlined in Table 2. The common repetitions vectors in the same scheduling PSM allows a common static schedule to be applied across all modes in that PSM. For example, for PSM_1, the static schedule $\sigma_1 = ABC$ can be applied as the schedule for H. Similarly, for all modes in PSM_2, we can apply the static schedule $\sigma_2 = AB(2C)$. Here, we apply looped scheduling notation, where a parenthesized term of the form (mX), where m is a non-negative integer (or a symbolic expression that resolves to a non-negative integer) and X is a sequence of actor firings, represents the successive execution m times of the sequence X. For background on the construction and manipulation

Table 1 Details of actor parameters, modes, and dataflow rates.

Actor	Configuration	Mode	Prod	Cons
A	N/A	A_0	0	(0,1)
	$p_1 = 0$	A_1	1	(2,0)
	$p_1 = 1$	A_2	1	(1,0)
B	N/A	B_0	0	(0,2)
	$(p_1, p_2) = (0, 0)$	B_1	2	(1,0)
	$(p_1, p_2) = (0, 1)$	B_2	2	(2,0)
	$(p_1, p_2) = (1, 0)$	B_3	1	(2,0)
	$(p_1, p_2) = (1, 1)$	B_4	1	(1,0)
C	N/A	C_0	0	(0,1)
	$p_2 = 0$	C_1	1	(2,0)
	$p_2 = 1$	C_2	1	(4,0)

Table 2 Scheduling PSMs of the hierarchical actor H.

PSMs	Mode of H	Mode of ABC	**q**
PSM_1	H_0	A_0 B_0 C_0	(1,1,1)
	H_1	A_1 B_1 C_1	(1,1,1)
	H_2	A_2 B_4 C_2	(1,1,1)
PSM_2	H_3	A_1 B_2 C_2	(1,1,2)
	H_4	A_2 B_3 C_1	(1,1,2)

of looped schedules for synchronous and parameterized dataflow graphs, we refer the reader to [4, 2].

For the entire application graph in this example, we can apply the schedule $\sigma_{top} = src\sigma_H(nsnk)$, where n is the mode-dependent firing rate (iteration count) for snk, and σ_H is configured dynamically as σ_1 or σ_2 based on the currently-active scheduling PSM.

We constructed the PSMs and schedules outlined here by hand, and based on these constructions, we implemented this synthetic application graph using the lightweight dataflow environment (LIDE), which is a tool for experimenting with dataflow techniques in arbitrary simulation- or platform-oriented languages, such as C, CUDA, MATLAB, and Verilog [19, 20]. Specifically, in our experiments we employed LIDE-C and LIDE-CUDA, which are C- and CUDA-oriented versions of the LIDE environment, respectively.

We implemented each actor as a simple sample rate converter that inserts or discards tokens to achieve the specified dataflow rates. The experiment is carried out using a desktop computer equipped with an Intel Core i7-2600K 8-core CPU, and 16GB memory. Figure 3 shows the execution time of the graph using CFDF canonical scheduling and PSM-level static scheduling. For our implementation of PSM-level static scheduling, we used the hierarchy of schedules σ_{top}, σ_1, and σ_2 defined above. In this example, the average execution time improvement of PSM-level static scheduling among the different modes of H is 11.9%.

Although it is based on a synthetic dataflow graph, the simplicity of this example helps to demonstrate concisely and concretely the proposed PSM-level static

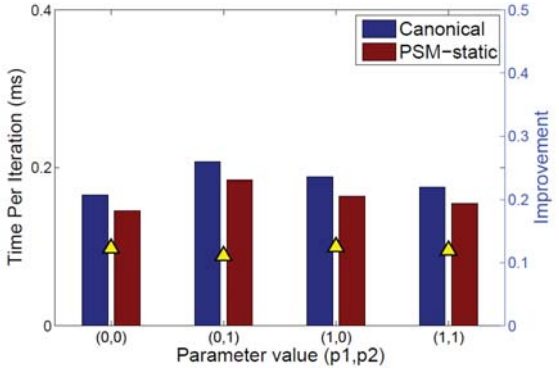

Fig. 3 Execution time comparison between canonical scheduling and PSM-level static scheduling for the synthetic example of Figure 2.

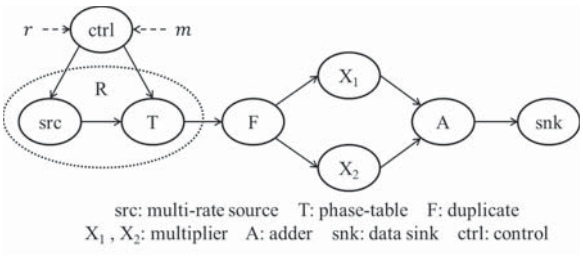

Fig. 4 A dynamically configurable RPSK modulator in CFDF.

scheduling approach, and the potential for performance improvement using the approach.

4.5 Application Example

In this section, we demonstrate a practical example of PSM-level static scheduling that is relevant to the cognitive radio domain. Figure 4 shows a dynamically configurable RPSK modulator that supports multiple source rates and multiple Phase-Shift-Keying (PSK) modulation schemes. The hierarchical actor R encompasses a subgraph that contains two CFDF actors *src* (using a minor abuse of notation), and T, whose modes are shown in Table 3. Here, r and m specify the source rate and the modulation scheme, respectively.

Using PSM-level static scheduling, we derive 4 PSMs, as shown in Table 4. The static schedule for each PSM is then constructed by hand, implemented in LIDE, and compared with canonical scheduling, as in Section 4.4. We see from the results that in this example, the performance improvement from applying PSM-level static scheduling is higher compared to that of the small, synthetic example in Section 4.4. In terms of the execution time per graph iteration (i.e., per minimal periodic scheduling iteration of the derived SDF subgraphs), PSM-level static scheduling outperforms canonical

Table 3 Dynamic actors in the RPSK application.

Actor	Mode	Prod	Cons
src	*INIT*	0	1
	src_1	1	0
	src_2	2	0
T	*INIT*	0	(0,1)
	BPSK	1	(1,0)
	QPSK	1	(2,0)
	$16-PSK$	1	(4,0)

Table 4 PSMs of the hierarchical actor R in the RPSK application.

PSM	Mode of R	Mode of *src* T	**q**
PSM_1	R_0	*INIT*,*INIT*	(1,1)
	R_1	src_1, *BPSK*	(1,1)
	R_2	src_2, *QPSK*	(1,1)
PSM_2	R_3	src_1 *QPSK*	(2,1)
	R_4	src_2, $16-PSK$	(2,1)
PSM_3	R_5	src_1 $16-PSK$	(4,1)
PSM_4	R_6	src_2 *BPSK*	(1,2)

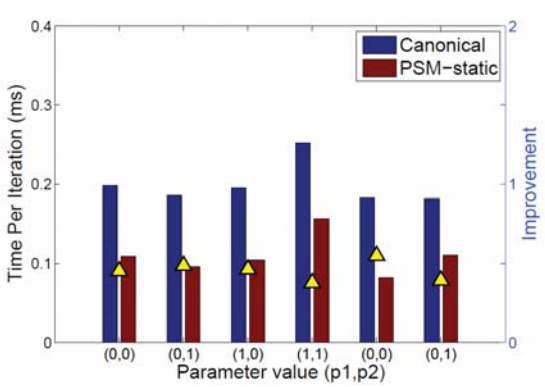

Fig. 5 Execution time comparison between canonical scheduling and PSM-level static scheduling for the RPSK application.

scheduling by an average of 45.4%, as shown in Figure 5. Here, the average is taken across the 6 operational modes for the hierarchical actor R.

4.6 Summary of PSM-level Static Scheduling

In this section, we have demonstrated a specific method, called PSM-level static scheduling, for applying the PSM modeling approach. There are many possible ways of applying PSMs in the design process, and the method presented in this section can be viewed as a specific way that we have studied and experimented with to help validate the utility of the PSM model. Although the PSM-level static

schedules experimented with in this section were constructed by hand, their foundation in the PSM and CFDF formalisms makes them amenable to derivation through general, automated techniques. Development of such automated tool support for PSM-level static scheduling and other applications of PSMs is a useful direction for further investigation.

5 PSM-level Actor Mapping on Heterogeneous Platforms

5.1 Overview

In this section, we demonstrate the application of PSMs to mapping actors in a CFDF-based dataflow program onto a heterogeneous platform. The targeted platform here consists of a general purpose CPU (called "host"), and a graphics processing unit (GPU) that is used to accelerate selected actors. The GPU is controlled by the host, and has a separate memory address space.

The execution of an actor in this environment on the GPU device generally involves three steps: host-to-device data transfer, on-device execution, and device-to-host data transfer. The data transfers between processors can result in significant overhead, which makes it unfavorable in some scenarios, such as when the amount of data to be processed is relatively small. Thus, the selection of actors to execute on the GPU (processor assignment) is an important problem for performance optimization.

We first formulate a general version of the processor assignment problem that is addressed in this section, and we describe our PSM-level processor selection approach in this general context. Then we present experimental results for PSM-level processor selection on the specific CPU-GPU heterogeneous platform described above.

5.2 PSM-level Processor Selection

Suppose that we have a CFDF graph $G = (V, E)$, and a target platform consisting of a (possibly heterogeneous) processor set $P = \{p_1, p_2, \ldots, p_n\}$. Also, for an actor A in G, let M_A denote the set of CFDF modes of A. The objective of PSM-level processor selection is to derive a set of PSMs and a "top-level" quasi-static schedule with the goal of optimizing a pre-defined performance metric. More specifically, PSM-level processor selection involves the following tasks:

- for each actor A, derivation of a set of n PSMs, $\mathit{selection}(A) = \nu(A,1), \nu(A,2), \ldots \nu(A,n)$, where each $\nu(A,i)$ represents the subset of modes in M_A that are to be assigned (during graph execution) to processor p_i;
- construction of a "top-level", quasi-static schedule that executes actors in G based on the dynamically-determined processor assignment defined by $\{\mathit{selection}(A)\} \mid A \in V$ together with the current parameter values (actor configurations) of the actors in V.

In our development of PSM-level processor selection in the remainder of this section, our targeted performance metric is throughput. However, the proposed

processor selection framework can be readily targeted to other metrics, such as latency or memory utilization or to composite metrics, such as latency-constrained throughput optimization, and memory-constrained latency optimization.

5.3 Profile-based Selection

In this section, we develop a profile-driven approach to PSM-level processor selection. We refer to this approach as *profile- and PSM-based processor selection* (*PAPPS*). In PAPPS, a three-dimensional "profile table" is used to characterize the performance of specific actor modes on specific processors. In particular, for a given mode $m \in M_A$ for an actor A, and a given processor $p \in P$, $profile(A, m, p)$ provides an estimate of the execution time of mode m for actor A on processor p. The profile table entries for a given actor can be obtained, for example, by iteratively (e.g., through appropriate simulation scripts) executing the actor on each processor in every mode and averaging the results for each mode.

After the profile table is constructed, PSMs for each actor A are formed by grouping together modes that perform best on a specific processor with ties being broken arbitrarily. Thus, for each actor A and each $i \in \{1, 2, \ldots n\}$, we have that

$$\nu(A, i) = \bigcup \{\{m\} \mid i = argmin_j(A, m, p_j)\}. \quad (4)$$

In the PAPPS approach, ties with respect to the *argmin* function in Equation 4 are resolved arbitrarily (as implied earlier), although more sophisticated schemes can be envisioned that take ties or "near-ties" (multiple alternatives that have competitive performance) into account in strategic ways. Such exploration of more sophisticated PSM-based processor assignment schemes is an interesting direction for further work.

Once the PSMs are constructed based on Equation 4, a top-level, quasi-static scheduler is used to visit actors according to some scheduling policy, and to execute each visited actor A using a target processor that is (dynamically) selected based on the currently-active PSM for A. In other words, each time an actor A is visited by the scheduler, the current mode m of A is examined to determine the active PSM (i.e., the unique $\nu(A, i)$ that contains m), and then processor p_i is selected as the processor on which to execute the next firing of A.

Canonical scheduling, described in Section 4, is a general policy that can be used as the top-level scheduling policy in this context. However, in some cases, static analysis of the parameterized application structure can be applied to streamline the policy — for example, by statically fixing the order of schedule traversal in a way that eliminates or greatly reduces the need for run-time enable condition checking. We demonstrate a simple example of such static-analysis-based streamlining in Section 5.4.

5.4 OFDM Demodulation

To demonstrate the PAPPS approach, we have applied it to an OFDM demodulator and a heterogeneous
CPU/GPU implementation platform, as described in Section 5.1. Orthogonal frequency division multiplexing (OFDM) is used extensively in high-speed wireless

communication systems because of its spectral efficiency, robustness in terms of multi-path propagation, and high bandwidth efficiency [8]. The OFDM demodulator is one of the fundamental subsystems of LTE and WiMAX wireless communication systems.

Figure 6 illustrates a runtime-reconfigurable OFDM demodulator that is modeled as a CFDF graph. Here, actor *SRC* represents a data source that generates random values to simulate a sampler. In a wideband OFDM system, information is encoded on a large number of carrier frequencies, forming an OFDM symbol stream. In baseband processing, a symbol stream can be viewed in terms of consecutive vectors of length N. The symbol is usually padded with a cyclic prefix (CP) of length L to reduce inter-symbol inference (ISI) [1]. In Figure 6, the CP is removed by actor *RCP*. Then, actor *FFT* performs a fast Fourier transform (FFT) to convert the symbol stream to the frequency domain.

In practical systems, further processing, such as frequency domain synchronization and channel estimation, is required to remove various channel effects. In this case study, however, we use a simpler design that directly performs symbol demapping to illustrate the PAPPS methodology. Actor *Demap* is a parameterized symbol demapper that performs M-ary QAM demodulation, with a configurable QPSK configuration ($M = 2$ or $M = 4$). The output bits are collected by the data sink (actor *SNK*).

For the targeted CPU/GPU platform described in Section 5.1, all of the actors in our OFDM demodulation system have CPU implementations, and some of the actors have GPU implementations.

Each actor A has a parameter, called the vectorization degree and denoted by $\beta(A)$, which is the number of OFDM symbols to be processed in a single activation (scheduler visit) of the actor. If the actor A is understood from context, then we sometimes drop the "(A)" and simply write β. Vectorization of signal processing dataflow graph actors, also referred to as "block processing", is useful in optimizing throughput, which is the targeted objective in our development of PSM-level processor selection (see Section 5.2) [17]. Here we assume that the same demapping scheme can be applied to all symbols to be processed in one activation, so that SIMD processing can be applied in vectorized executions.

In addition to β, actors in this design have a parameter M, which prescribes the number of bits per symbol. For example, if $M = 4$ and $\beta = 10$, this means that the system is operating in a mode that uses QAM16 as the demapping scheme, and executes actors in blocks of 10 firings each. A third actor parameter is the OFDM symbol length, which we denote by N.

The parameter values in this example determine the mode of each actor, and the actor mode determines the production and consumption rates. Note that this is not always the case in CFDF actors, where, for example, the next mode for an actor can be different from the current mode even though there is no change in parameter settings (e.g., see [15]). However, because there is no such dynamics involved with next mode determination in this example, the actors can be mapped into corresponding parameterized synchronous dataflow (PSDF) actors [2]. The example, therefore demonstrates the applicability to PSM techniques to PSDF graphs.

Table 5 shows the valid parameter values for the actors in our OFDM demodulation system. The mode set of *Demap* is given by Equation 3 in Section 3.

Table 5 Actor parameters in the OFDM demodulator system.

Parameter	Domain
β	$\{1, 10, \ldots, 100\}$
N	$\{512, 1024\}$
M	$\{2, 4\}$

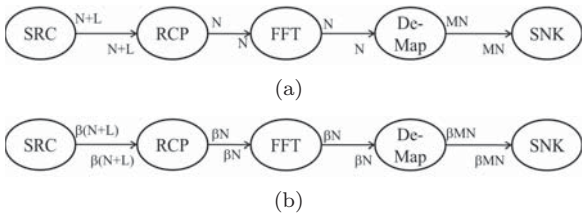

Fig. 6 PSM-CFDF model of a configurable OFDM demodulator. (a) Original dataflow graph. (b) Vectorized dataflow graph.

Similarly, for other actors, valid combinations of parameter values lead uniquely to their mode settings. These details for the other actors are omitted here for brevity.

5.5 Application of PAPPS to the OFDM Demodulation System

The PSM-CFDF actors *RCP*, *FFT* and *Demap* are each implemented on both the CPU and GPU processors. Following the profiling approach described in Section 5.3, each actor A is profiled in every mode in its mode set M_A for both the CPU and GPU implementations. The results are then used to construct the profile table *profile*.

In our experiments, an NVIDIA GTX680 GPU with 2GB memory and an Intel Core I7 3.4GHz CPU with 8GB memory are used for GPU implementation and CPU implementation, respectively. Figure 7 illustrates the profile table *profile* for the actors. The maximum latency for all vectorization degrees considered is less than 8 ms, which is tolerable in many software defined radio contexts. In the case of *RCP*, which removes the cyclic prefix from the received signal, the CPU implementation performs better in all settings. This is due to the small amount of computation performed in this actor compared to the large CPU-GPU memory transfer overhead. As a result, *selection*(*RCP*) contains only one non-empty PSM; the PSM associated with the GPU has no modes.

For the *FFT* actor, the GPU implementation always performs better than the CPU implementation in the same mode. Thus, for this actor, the PSM associated with the CPU has no modes. For the *Demap* actor in the 16-QAM modes ($M = 4$), the GPU implementation outperforms the CPU implementation for all values of the vectorization degree β. In the QPSK modes ($M = 2$), there is less difference in performance, and the CPU implementation generally performs better for lower β values, while the GPU implementation performs better for higher β values. The smaller computational load in the QPSK modes makes the memory transfer overhead more significant, which leads to a smaller performance gain from the

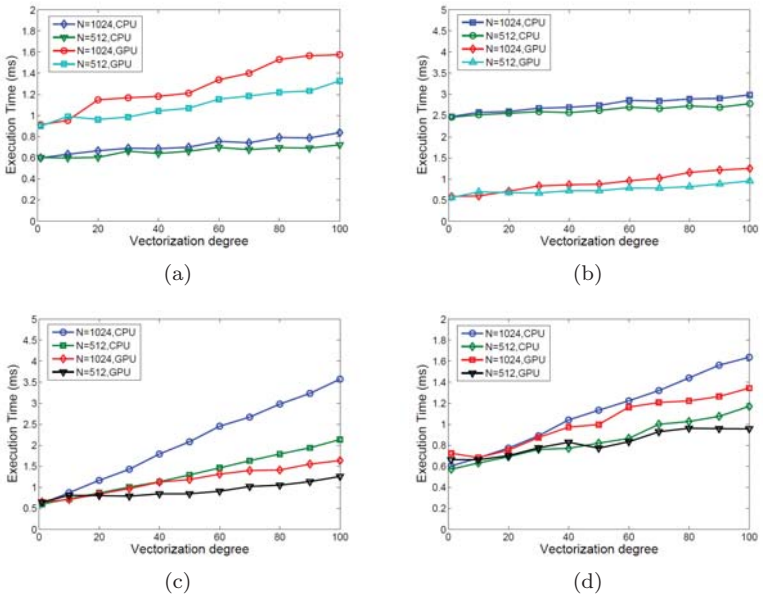

Fig. 7 Actor profiles for application of PAPPS to the OFDM demodulator: (a) *RCP* actor; (b) *FFT* actor; (c) *Demap* actor in 16-QAM modes; (d) *Demap* actor in QPSK modes.

Table 6 PSM grouping based on CPU and GPU performance profiles for processor selection. PSM_1 and PSM_2 are the sets of modes that have shorter execution times for CPU- and GPU-based execution, respectively.

Actor	PSM_1	PSM_2
RCP	$N = 512, 1024;\ 1 \leq \beta \leq 100$	$\varnothing$
FFT	$\varnothing$	$N = 512, 1024;\ 1 \leq \beta \leq 100$
Demap	$\{N = 1024, M = 4,\ \beta = 1\}$ $\{N = 512, M = 4,\ \beta = 1, 10\}$ $\{N = 1024, M = 2,\ \beta = 1, 10\}$ $\{N = 512, M = 2,\ 1 \leq \beta \leq 40\}$	$\{N = 1024, M = 4,\ 10 \leq \beta \leq 100\}$ $\{N = 512, M = 4,\ 20 \leq \beta \leq 100\}$ $\{N = 1024, M = 2,\ 20 \leq \beta \leq 100\}$ $\{N = 512, M = 2,\ 50 \leq \beta \leq 100\}$

GPU. In summary, the *Demap* actor has two non-empty PSMs $\nu(Demap, p_1)$ and $\nu(Demap, p_2)$.

Table 6 shows the grouping of actor modes into PSMs when applying the PAPPS method based on the achieved profiling results illustrated in Figure 7.

We have implemented the OFDM demodulator system on the targeted CPU/GPU platform using a PAPPS-based processor selection scheme based on the PSMs illustrated in Table 6. We streamlined the top-level scheduler (see Section 5.3) by observing that even though the production and consumption rates of actors can vary based on the active actor modes, the variations in this application are interdependent such that the dataflow graph exhibits SDF behavior, and furthermore, the repetitions vector remains constant. In particular, the repetitions vector is specified by $q(A) = 1$ for each actor A regardless of what actor modes are active. This allows us to implement the top-level scheduler without any run-

time checks for actor enabling conditions. Note, however, that even though SDF techniques are employed, the derived scheduler should not be viewed as a form of static scheduling because the processor assignment can change dynamically.

As in the case study of Section 4, we implemented the top-level scheduler by hand. This scheduler implementation incorporates the PAPPS method for dynamic processor selection based on the PSM decompositions illustrated in Table 6. Building on the developments of this section to construct automated scheduler derivation for PAPPS-based implementation is an interesting direction for future work.

5.6 Experimental Results

We compared the application throughput of alternative implementations in terms of the execution time per (vectorized) application iteration, where an application iteration in this context corresponds to the processing required for $(\beta \times N)$ symbols of the enclosing OFDM system. Because we compare alternative processor selection schemes with β fixed for each comparison point, this method of throughput comparison does not favor any particular kind of scheme.

Figure 8 shows the execution time per application iteration for three types of processor selection schemes: (1) all actors are assigned to the CPU ("CPU"), (2) *RCP*, *FFT* and *Demap*, the most computationally-
intensive actors, are assigned to the GPU ("GPU"), and (3) processor selection is performed dynamically using our implemented PAPPS-based scheduler ("PAPPS"). The speedups achieved by using PAPPS, compared to methods (1) and (2), are also shown in the figure. The average speedup achieved by PAPPS in this application over a CPU implementation is more than 1.5X. In the setting where the largest amount of data is present (1024-FFT and 16-QAM), the average speedup is more than 2X over all vectorization degrees. The achieved speedup is limited by the cost of data transfer between CPU and GPU memory for each actor. This data transfer overhead has been taken into account in the reported speedup values.

Compared to the GPU implementation scheme
(scheme (1)), the PAPPS scheme achieves an average of 20% improvement in throughput over the GPU scheme. However, the vectorization step applied in our implementation generally results in increased latency for the system. In wireless communication applications, latency is a critical design constraint, and thus, vectorization should be applied carefully to ensure that excessive latency does not result.

In our experiments, the vectorization degree is set to be no more than 100. As shown in Figure 8, this results in a maximum latency of 8ms, which is reached when $N = 1024$ and $\beta = 100$. This is at a tolerable level of latency for many kinds of software radio systems. For example, 8ms is only a small fraction of the typical 250ms end-to-end delay for data packets, which is described for the communication systems discussed in [6]. In cases where there are more stringent latency constraints, the vectorization degree can be bounded more tightly to trade off throughput performance for decreased latency.

The experiments presented in this section along with the other examples discussed in this paper are provided to give a concrete idea of the kind of approaches

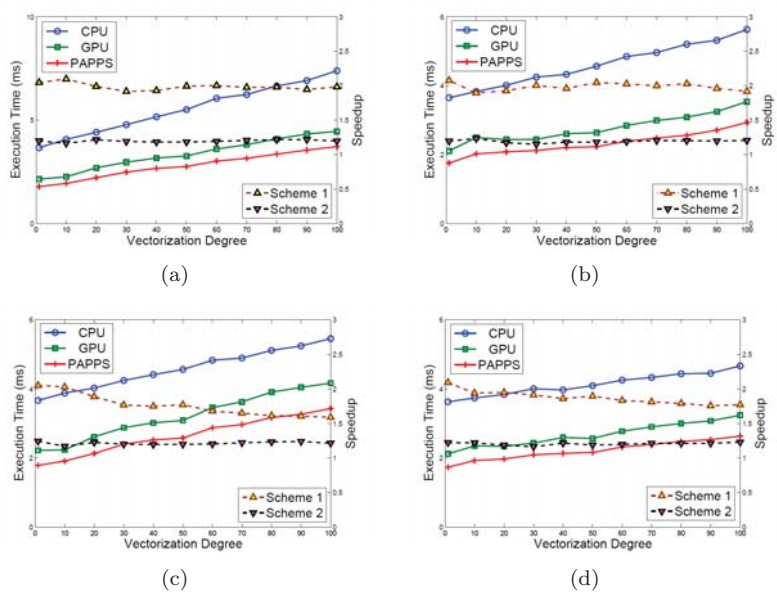

Fig. 8 Execution time and speedup under three types of processor selection schemes for the OFDM demodulator system: (a) 1024-pt FFT, 16-QAM; (b) 512-pt FFT, 16-QAM; (c) 1024-pt FFT, QPSK; (d) 512-pt FFT, QPSK. Solid lines represent execution times while dashed lines represent the speedup obtained by using the PAPPS approach. The brown dashed line with an "up-triangle" represents the speedup of PAPPS over CPU (scheme (1)); the black dashed line represents the speedup of PAPPS over GPU (scheme (2)).

that are supported by the PSM framework. These can be viewed as representative examples that help to give a sense of the diverse possibilities for applying the proposed methods. Further study into applying these methods and developing design optimizations that build on them is a useful direction for future investigation.

6 Conclusions

In this paper, we have introduced a new dataflow modeling technique called parameterized set of modes (PSM) and demonstrated its relevance and application to design and implementation of signal processing systems for cognitive radio applications. PSMs enable novel ways for representing, manipulating and applying related
groups of actor modes that lead to more concise formulations of actor behavior, and a unified modeling methodology for applying a variety of techniques for efficient implementation. To demonstrate the utility and versatility of PSMs in signal processing system design processes, we have developed two case studies involving mapping of important kinds of reconfigurable wireless communication subsystems into efficient implementations. The PSM methods introduced in this paper allow implementation techniques like those introduced in the case studies to be developed according to a common modeling framework, which allows such techniques to be better understood, integrated, and optimized. Several useful directions for

future work have also emerged from the developments of this paper, including the investigation of automated techniques for applying PSMs to efficient static region derivation and to processor selection on heterogeneous platforms.

7 Acknowledgement

This work was supported in part by the US National Science Foundation under grants CNS–1265332 and CNS–1264486; Tekes — the Finnish Funding Agency for Technology and Innovation — through the WiFiUS program; and the Laboratory for Telecommunications Sciences.

We are also grateful the anonymous reviewers for their constructive comments, which have helped significantly to improve the paper.

References

1. van de Beek, J.J., Sandell, M., Isaksson, M., Borjesson, P.O.: Low-complex frame synchronization in OFDM systems. In: Proceedings of the International Conference on Universal Personal Communications, pp. 982–986 (1995)
2. Bhattacharya, B., Bhattacharyya, S.S.: Parameterized dataflow modeling for DSP systems. IEEE Transactions on Signal Processing **49**(10), 2408–2421 (2001). DOI:10.1109/78.950795
3. Bhattacharyya, S.S., Deprettere, E., Leupers, R., Takala, J. (eds.): Handbook of Signal Processing Systems, second edn. Springer (2013). URL http://dx.doi.org/10.1007/978-1-4614-6859-2. ISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online)
4. Bhattacharyya, S.S., Murthy, P.K., Lee, E.A.: Synthesis of embedded software from synchronous dataflow specifications. Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology **21**(2), 151–166 (1999)
5. Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J.A.: Cyclo-static dataflow. IEEE Transactions on Signal Processing **44**(2), 397–408 (1996)
6. Blajić, T., Nogulić, D., Družijanić, M.: Latency improvements in 3G long term evolution. In: Proceedings of the International Convention on Information and Communication Technology, Electronics and Microelectronics (2006)
7. Buck, J.T., Lee, E.A.: Scheduling dynamic dataflow graphs using the token flow model. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (1993)
8. Edfors, O., Sandell, M., van de Beek, J.J., Landstrom, D., Sjoberg, F.: An introduction to orthogonal frequency division multiplexing. Tech. rep., Lulea University of Technology, Sweden (1996)
9. Eker, J., Janneck, J.W.: CAL language report, language version 1.0 — document edition 1. Tech. Rep. UCB/ERL M03/48, Electronics Research Laboratory, University of California at Berkeley (2003)
10. Falk, J., Zebelein, C., Haubelt, C., Teich, J.: A rule-based quasi-static scheduling approach for static islands in dynamic dataflow graphs. ACM Transactions on Embedded Computing Systems **12**(3) (2013)
11. Gu, R., Janneck, J., Raulet, M., Bhattacharyya, S.S.: Exploiting statically schedulable regions in dataflow programs. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, pp. 565–568. Taipei, Taiwan (2009)
12. Haubelt, C., Falk, J., Keinert, J., Schlichter, T., Streubühr, M., Deyhle, A., Hadert, A., Teich, J.: A SystemC-based design methodology for digital signal processing systems. EURASIP Journal on Embedded Systems **2007**, Article ID 47,580, 22 pages (2007)
13. Lee, E.A., Messerschmitt, D.G.: Synchronous dataflow. Proceedings of the IEEE **75**(9), 1235–1245 (1987)
14. Plishker, W., Sane, N., Bhattacharyya, S.S.: Mode grouping for more effective generalized scheduling of dynamic dataflow applications. In: Proceedings of the Design Automation Conference, pp. 923–926. San Francisco (2009)

15. Plishker, W., Sane, N., Kiemb, M., Anand, K., Bhattacharyya, S.S.: Functional DIF for rapid prototyping. In: Proceedings of the International Symposium on Rapid System Prototyping, pp. 17–23. Monterey, California (2008)
16. Plishker, W., Sane, N., Kiemb, M., Bhattacharyya, S.S.: Heterogeneous design in functional DIF. In: Proceedings of the International Workshop on Systems, Architectures, Modeling, and Simulation, pp. 157–166. Samos, Greece (2008)
17. Ritz, S., Pankert, M., Meyr, H.: Optimum vectorization of scalable synchronous dataflow graphs. In: Proceedings of the International Conference on Application Specific Array Processors (1993)
18. Sane, N., Hsu, C., Pino, J.L., Bhattacharyya, S.S.: Simulating dynamic communication systems using the core functional dataflow model. In: Proceedings of the International Conference on Acoustics, Speech, and Signal Processing, pp. 1538–1541. Dallas, Texas (2010)
19. Shen, C., Plishker, W., Wu, H., Bhattacharyya, S.S.: A lightweight dataflow approach for design and implementation of SDR systems. In: Proceedings of the Wireless Innovation Conference and Product Exposition, pp. 640–645. Washington DC, USA (2010)
20. Shen, C., Wang, L., Cho, I., Kim, S., Won, S., Plishker, W., Bhattacharyya, S.S.: The DSPCAD lightweight dataflow environment: Introduction to LIDE version 0.1. Tech. Rep. UMIACS-TR-2011-17, Institute for Advanced Computer Studies, University of Maryland at College Park (2011). Http://hdl.handle.net/1903/12147
21. Siyoum, F., Geilen, M., Moreira, O., Nas, R., Corporaal, H.: Analyzing synchronous dataflow scenarios for dynamic software-defined radio applications. In: Proceedings of the International Symposium on System-on-Chip, pp. 14–21 (2011)