Parameterized Core Functional Dataflow Modeling and Its Application to Wireless Communication Systems

by

Lai-Huei Wang

Table of Contents

Abstract

Due to the increased complexity of dynamics in modern DSP applications, dataflow-based design methodologies require significant enhancements in modeling and scheduling techniques to provide for efficient and flexible handling of dynamic behavior. In this report, we address this problem through a new framework that is based on integrating two complementary modeling techniques, core functional dataflow (CFDF) and parameterized synchronous dataflow (PSDF). We apply, in a systematically integrated way, the structured mode-based dynamic dataflow modeling capability of CFDF together with the features of PSDF for dynamic parameter reconfiguration and quasi-static scheduling. We refer to this integrated methodology for mode- and dynamic-parameter-based modeling and scheduling as core functional parameterized synchronous dataflow (CF-PSDF). Through a wireless communication case study involving MIMO detection, we demonstrate the utility of design and implementation using CF-PSDF graphs. Experimental results on this case study demonstrate the efficiency and flexibility of our proposed new CF-PSDF based design methodology.

1 Introduction

Dataflow models are widely used for expressing the functionality of digital signal processing (DSP) applications, such as those associated with audio and video data stream processing, digital communications, and image processing (e.g., see [1]). Their graph-based formalisms allow intuitive and yet semantically rigorous descriptions of applications. Such a semantic foundation enables a variety of analysis tools, including tools for determining buffer bounds and efficient scheduling (e.g., see [2, 3, 4]).

Due to the increased complexity of dynamics in modern DSP applications, such as wireless communication systems based on LTE and WiMAX, designers need significant flexibility in the types of functional behaviors that they can efficiently specify and implement. To model complex dynamic DSP systems, a variety of dataflow approaches have been proposed (e.g., see [1]). Some of these can model arbitrary dynamic behaviors, but may lead to inefficient schedules. Others allow powerful scheduling and mapping techniques by restricting the range of dynamic applications that they can accommodate.

Core functional dataflow (CFDF), is a dynamic dataflow model that provides highly expressive semantics for the design of applications with structured dynamic behavior [5]. However, this flexibility, especially when high levels of data-dependent dynamics are present, may result in significant run-time scheduling overhead and reduced predictability in scheduling performance.

On the other hand, *parameterized synchronous dataflow* (*PSDF*) is a modeling

technique that provides for systematic integration of dynamic parameter reconfiguration into synchronous dataflow representations [3]. Such an approach enables flexible parameterized modeling as well as strong support for quasi-static scheduling, which allows efficient and predictable scheduling performance for many kinds of dynamic applications. Here, by *quasi-static scheduling*, we mean scheduling techniques that fix a significant portion of schedule structure at compile time, while allowing flexibility for run-time adaptation of this statically constructed structure based on characteristics of input data and operating conditions [2]. To provide support for powerful quasi-static scheduling techniques, expression of dynamics in PSDF is restricted — in particular, dynamic changes to actor and subsystem dataflow properties are disallowed for some kinds of modeling structures [3].

This report is written and organized based on the work published in [6] with minimal changes required. In this report, we develop a new dataflow modeling framework, which is based on careful integration of the CFDF and PSDF models. We refer to our proposed model as *core functional parameterized synchronous dataflow* (*CF-PSDF*). CF-PSDF provides useful trade-offs among dynamic modeling flexibility, and support for efficient quasi-static scheduling. By applying our proposed design methodology based on CF-PSDF modeling, designers can potentially enhance performance of dynamic applications by employing efficient static and quasi-static scheduling techniques locally, and reducing the overhead associated with more general dynamic scheduling strategies. We demonstrate the utility of our proposed CF-PSDF based modeling and design techniques using an application case study involving MIMO detection.

2 Background and Related Work

2.1 Background

Parameterized dataflow is a meta-modeling technique that can significantly improve the expressive power of an arbitrary dataflow model that possesses a well-defined concept of a graph iteration [3]. Parameterized dataflow provides a method to systematically integrate dynamic parameter reconfiguration into such models of computation, while preserving many of the properties and intuitive characteristics of the original models. The integration of the parameterized dataflow meta-model with *synchronous dataflow* (*SDF*) provides the model of computation referred to as *parameterized synchronous dataflow* (*PSDF*). PSDF offers valuable properties in terms of modeling systems with dynamic parameters, supporting efficient scheduling techniques, and natural integration with popular SDF modeling techniques [3].

A PSDF specification (subsystem) is composed of three cooperating PSDF graphs, the *init*, *subinit*, and *body* graphs of the specification. The init graph is designed to configure the corresponding subinit and body graphs while the subinit graph can only change parameters in the body graph. The body graph, when executed, performs the main functionality of the subsystem based on the updated set of parameters. For more details on PSDF modeling, we refer the reader to [3].

Core functional dataflow (*CFDF*), a deterministic sub-class of *enable-invoke dataflow* (*EIDF*), is a dynamic dataflow model that provides highly expressive semantics for the design of applications with structured dynamic behavior [5]. In the formalism, each CFDF actor has a set of *modes*. When a given actor mode is exe-

cuted (invoked), an actor consumes and produces a fixed number of tokens. However different modes of the same actor can produce and consume different numbers of tokens, thereby allowing for actor-level dynamic dataflow behavior. Each actor has an *enabling function*, which indicates if a given mode may be executed given the present state of the application. The invoking function for a CFDF actor takes an enabled mode as an argument, executes the associated actor in that mode, and then determines the next mode in which the actor will be executed (after it has been enabled and subsequently invoked).

2.2 Related Work

In addition to CFDF and PSDF, there is a variety of other models that support dynamic dataflow modeling, design, and implementation. Wiggers, Bekooij, and Smit [7] present *variable rate dataflow* (*VRDF*) to model systems with data-dependent communication, and develop techniques to compute buffer sizes for VRDF specifications for given throughput constraints. Eker et al. [8] present a hierarchical approach for modeling of heterogeneous embedded systems, including systems that incorporate dataflow behaviors. In this approach, designers employ modeling constructs called *directors* to control the communication and execution schedules for associated application subsystems. The *stream-based functions* (*SBF*) model of computation combines the semantics of dataflow and process network models for design and implementation of embedded signal processing systems [9]. An actor in SBF contains a set of operational functions, along with a controller, state, and a

transition function. The use of operational functions and the transition function in SBF is analogous in some ways to the modes and next mode determination functionality in the CFDF model. Given this relationship, an interesting direction for further study is the adaptation of the techniques introduced in this work to SBF specifications (i.e., an integrated SBF-PSDF modeling framework).

The dataflow-based modeling and design techniques presented in this report differ from the related work discussed above in that our framework generalizes CFDF and PSDF models to provide systematic mode-based dynamic modeling together with flexible dynamic parameter reconfiguration. Our emphasis on support for localized use of optimized static and quasi-static schedules further distinguishes our contribution in this report from related work in this area.

The work presented in this report is to be presented also at the 2013 IEEE Workshop on Signal Processing Systems [6].

3 CF-PSDF Modeling and Scheduling

In CF-PSDF, a DSP application is modeled through a *CF-PSDF specification*, which is also called a *CF-PSDF subsystem*. A hierarchical actor that encapsulates a CF-PSDF subsystem S (i.e., for instantiation in a higher level subsystem) is called the *CF-PSDF actor* associated with subsystem S. A CF-PSDF actor H can be viewed at its interface as a CFDF actor that has a set of modes, and *enable* and *invoke* functions, which are fundamental components of the CFDF model [5]. When H is executed in a given mode, a fixed number of tokens is consumed and produced

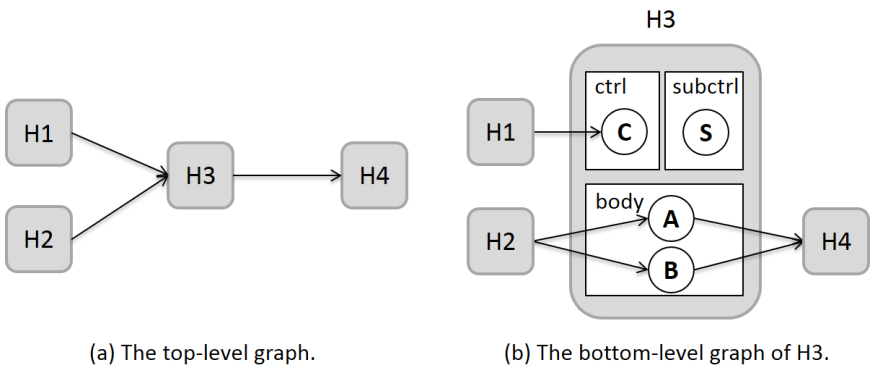

Figure 1: An example of a CF-PSDF graph.

at the input and output ports of H, respectively. Across different modes of H, however, the production and consumption rates at the ports of H can vary.

In a CF-PSDF actor H, the encapsulated specification, which we denote by $\sigma(H)$, is decomposed into three cooperating graphs, which we refer to as the *ctrl* (ϕ_c), *subctrl* (ϕ_s), and *body* (ϕ_b) graphs of $\sigma(H)$ (here, "ctrl" is used as an abbreviation for "control"). The actor `H3` in Figure 1(b) shows an example of a CF-PSDF actor.

As in PSDF modeling, the body graph of a CF-PSDF specification is intended for use in modeling the core functional behavior of the associated subsystem, while the ctrl and subctrl graphs, which are analogous in some ways to the init and subinit graphs of PSDF, control the dynamic behavior of the body graph. This dynamic body graph control is achieved by appropriately configuring selected body graph parameters. As in PSDF, the subctrl graph of a CF-PSDF specification $\sigma(H)$ can configure the parameters in the associated body graph in ways that do not change the production and consumption rates at the interfaces (ports) of H.

The ctrl graph of a CF-PSDF subsystem $\sigma(H)$ is executed once during each

firing of H and is allowed to update parameters in the associated subctrl and body graphs. Such parameter configurations may depend on parameters of the enclosing system as well as on run-time data generated from other CF-PSDF actors (i.e., data-dependent parameter updates).

On specific parameter that is configured in the ctrl graph of $\sigma(H)$ is a special parameter $\mu(H)$ that controls the execution mode of H. The ctrl graph is the basic mechanism in CF-PSDF for determining this execution mode. After the ctrl graph of $\sigma(H)$ executes and $\mu(H)$ is updated, the production and consumption rates at the interfaces of H are fixed until the next execution of the ctrl graph. Furthermore, the control information processed in the ctrl graph of $\sigma(H)$ can be shared by "exporting" tokens (through dedicated dataflow graph edges) to ctrl graphs in other CF-PSDF actors within the enclosing application model. This mechanism of "sharing" control information, which represents a departure from the parameterized dataflow meta-model, can facilitate local scheduling and mapping optimization, and help avoid repetitive computation of control information. Additionally, the ctrl graph can process data from input ports of $\sigma(H)$ and produce data onto the output ports of $\sigma(H)$. This is more flexible compared to the init graph of PSDF, where such linkages to the ports of the enclosing PSDF actor are not allowed.

The modeling flexibility of CF-PSDF compared to PSDF is illustrated in Figure 2. In the PSDF subsystem shown in Figure 2, the production rate of actor A must be independent of the output of actor X. However, this kind of data-dependent dynamics can be useful to model when developing DSP applications. For example, in wireless communications, a turbo decoders with a dynamic iteration count may or

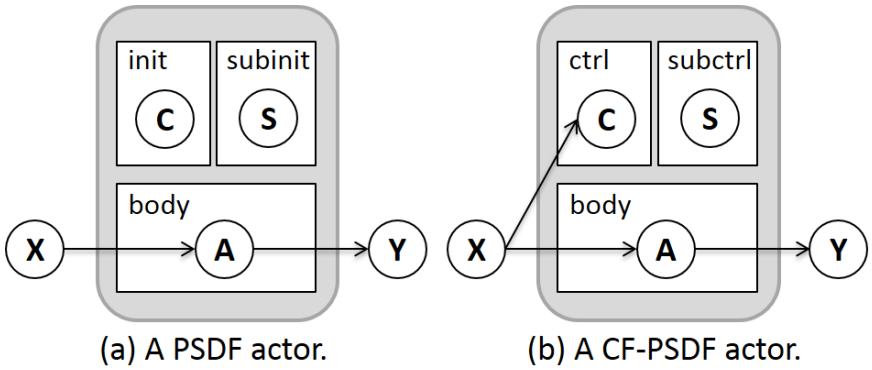

Figure 2: Examples of PSDF and CF-PSDF actors.

may not execute one more iteration according to the run-time output of the current iteration [10]. Another example of this kind of dataflow dynamics is discussed in Section 4.

On the other hand, in CF-PSDF, designers can pass control tokens from actor X to the ctrl graph of a CF-PSDF subsystem, as shown in Figure 2(b). Then, the ctrl graph can configure the dataflow (production and consumption) rates of A through the CF-PSDF mechanism of parameter reconfiguration based on subsystem input tokens (input tokens arriving from actor X in this case). A disadvantage in supporting this kind of dynamics is that the efficient quasi-static scheduling techniques that have been developed for PSDF models (e.g., see [3]) are in general not applicable to CF-PSDF specifications. However, in Section 3.3, we develop new scheduling techniques that exploit the structure of CF-PSDF models, and permit derivation of efficient schedules from such models.

3.1 Multi-Mode Actors

In this section, we develop scheduling techniques for mapping CF-PSDF graphs into efficient implementations.

The hierarchical, mode-oriented structure of CF-PSDF modeling allows designers to specify complex applications with more concise graphs representations, where related functionality can be grouped together naturally under common actors or subsystems. For example, a P-QAM mapper, which maps blocks of $log_2 P$ input bits to P-QAM symbols, consumes P tokens (bits) and produces one token (QAM symbol). An SDF representation of this functionality would typically require three separate actors for 4-QAM, 16-QAM, and 64-QAM processing, while this entire functionality can be encapsulated within a single, multi-mode CFDF actor. However, in a CFDF graph, additional control modes may be needed to provide for correct transitioning between operational states (e.g., see [5]), which may increase design effort and introduce scheduling overhead. In CF-PSDF, we alleviate these problems by applying a central control mechanism, through ctrl and subctrl graph execution, and a modeling approach based on designer-specified sets of practical mode combinations (functional modes) across body graph actors. This concept of functional modes is discussed next, in Section 3.2.

3.2 Subsystem Modes

In CF-PSDF, it is not possible for one body graph actor to have direct control over the dataflow rates of another actor in the same body graph. For example, in

the graph of Figure 1(b), the dataflow rates of actors A and B can be varied based on output from `H1`, but the dataflow rates of A are prohibited from depending on outputs of B, and vice versa. This condition ensures that the mode transitions of all actors in a body graph ϕ_b can be configured centrally from the ctrl graph based on system parameters and run-time data. This centralized, ctrl-graph based control of actor modes can help to eliminate certain local (actor-level) modes and transitions that are employed in pure CFDF models to ensure proper transitioning between processing states.

Each CF-PSDF actor (subsystem) H has a special mode called the *control mode* of H, which is used to execute the ctrl graph of H, and update parameters, including the next mode parameter $\mu(H)$. The control mode can in general consume and produce data at the interface ports of H, as described previously.

Apart from the control mode, a CF-PSDF actor H may have any number of additional modes, which are referred to as the *functional modes* of H. Execution of H proceeds based on alternating sequences of the control mode and a functional mode (i.e., between each pair of successive functional mode executions, there is exactly one execution of the control mode). Each functional mode of H corresponds to a unique set of modes for all actors that are contained in the associated body graph, ϕ_b. That is, for each functional mode m of H and each actor α in ϕ_b, there is a unique mode $z(m, \alpha)$ of α that governs the execution state of α whenever H executes in functional mode m. Thus, in each functional mode m, ϕ_b can be viewed an SDF graph $G_{sdf}(m)$, which can be analyzed and scheduled by leveraging the large body of existing techniques for SDF (e.g., see [1]).

Note that in CF-PSDF, the set $F(H)$ of functional modes of H is defined explicitly by the designer. An alternative approach would be to derive $F(H)$ by enumerating all possible mode combinations across the actors within ϕ_b. However, this approach is clearly not scalable since, for example, there is no polynomial bound on such mode combinations.

Indeed, in practical DSP applications, many mode combinations may be uninteresting (e.g., redundant or simply not useful). In Figure 1(b), for example, suppose that actors A and B are both P-QAM mappers with three modes each for $P = 4$, 16, and 64. The set of all mode combinations for `H3` contains $3 \times 3 = 9$ combinations. However, at any given time during actual execution of the system, the values of P will be identical for both A and B — only three mode combinations are relevant in the design of `H3`. Thus, `H3` is designed such that $F(H3)$ contains only three modes.

In previous work, methods have been developed to detect and eliminate unreachable mode combinations in CFDF graphs [5]. However, in practical scenarios, such as the example of Figure 1(b), it can be difficult to detect all unused modes without designer guidance. Automated techniques, such as those developed in [5], can be used in a complementary fashion to the designer-specified approach in CF-PSDF (e.g., to remove unused modes from the specified functional mode set). Integrating such automation into the CF-PSDF framework is an interesting direction for future work.

3.3 Scheduling Techniques

In CF-PSDF, dynamically parameterized and dynamic dataflow subsystems are represented with two-level hierarchies, as illustrated in Figure 1. In the top level (e.g., Figure 1(a)), each actor is a CF-PSDF actor with associated enable and invoke functions, which have similar roles as in the pure CFDF model. The lower level of the subsystem design hierarchy (e.g., Figure 1(b)) is composed of three subgraphs to provide flexible dynamic parameter reconfiguration. This structured decomposition into three subgraphs is based on a similar kind of decomposition provided in PSDF, but with significant adaptations to make the modeling approach more flexible and more coupled to CFDF design techniques.

CF-PSDF provides a natural framework for quasi-static scheduling based on the decomposition of a CF-PSDF subsystem H in terms of it functional modes $F(H)$ and the associated set of SDF graphs

$$S(H) = \{G_{sdf}(m) \mid m \in F(H)\} \tag{1}$$

that characterizes the body graph ϕ_b. Each graph $\{R \in S(H)\}$ can be scheduled using SDF techniques, and based on specific operational constraints (e.g., constraints on throughput, latency, or buffer memory requirements) that are associated with the corresponding functional mode of H. The resulting set of SDF schedules $S(R) \mid R \in S(H)$ can then be integrated in a quasi-static, dynamic control-driven manner using CFDF techniques for scheduling H as a component within its enclosing subsystem or application graph model.

For example, for the dataflow graph G_{outer} that contains H, one can readily apply a *CFDF canonical schedule*, which is a standard type of schedule for CFDF graphs that can be constructed quickly and is suitable for rapid prototyping and bottleneck identification [5]. Alternatively, existing techniques for CFDF schedule optimization (e.g., see [5]) can be applied to G_{outer} to help improve system performance or satisfy operational constraints.

4 Case Study

In this section, we demonstrate the utility CF-PSDF-based implementation with a case study of soft multiple-input-multiple output (MIMO) detection.

4.1 Application Model based on CF-PSDF

MIMO technology has been adopted in many modern wireless communication standards, such as LTE and WiMAX, due to the significant capacity increases that can be achieved by using multiple antennas in transmitters and receivers [11]. In this case study, we implement an application of $M \times M$ MIMO detection with a P-QAM constellation. We apply an efficient soft MIMO detection algorithm called the *list fixed-complexity sphere decoder* (LFSD), which is a list-based version of the fixed-complexity sphere decoder (FSD) [12]. The LFSD generates a list of candidates around the maximum likelihood (ML) solution that can be used to calculate soft-output information for each transmitted bit b_k in the form of log-likelihoods (LLRs), $\{L_k\}$.

An $M \times M$ MIMO system is commonly decomposed into M processing layers (in our experiments, we use $M = 4$). In our design, the vector-valued parameter λ, consisting of M-elements, specifies the number of optimal detected results that are generated at each layer. If $\lambda = (n_1, n_2, \ldots, n_M)$, then the list size can be expressed as $N_L = \prod_{i=1}^{M} n_i$.

In our implementation, the soft MIMO detector takes the received symbol vector y and the channel matrix C as inputs, finds the N_L candidates for each y and C, and generates the soft information L_k. We model the application with our proposed CF-PSDF framework, as illustrated in Figure 3. Here, the dataflow graph is composed of three CF-PSDF actors (`Pre-processor`, `LFSD`, and `Post-processor`), two source actors `Y` (source of y) and `H` (source of C), and one sink actor `B` (sink of L_k). The soft MIMO detector is divided into three parts: (1) the preprocessing component (`Pre-processor` actor), which applies QR decomposition on the channel and least squares estimation of the input symbols; (2) the LFSD component (`LFSD` actor), which generates a list of candidates according to the FSD algorithm; and (3) the postprocessing component (`Post-processor` actor), which computes the LLRs with the list generated by the LFSD component. On the subsystem corresponding to each CF-PSDF actor, the quasi-static scheduling technique developed in [3] is applied.

In our implementation, the list size N_L is determined dynamically for each realization (i.e., for each y and C) based on the channel quality. Usually, a large value for N_L improves the bit error rate (BER), but at the cost of increased complexity. In our design, a realization with better channel quality is processed with a smaller list

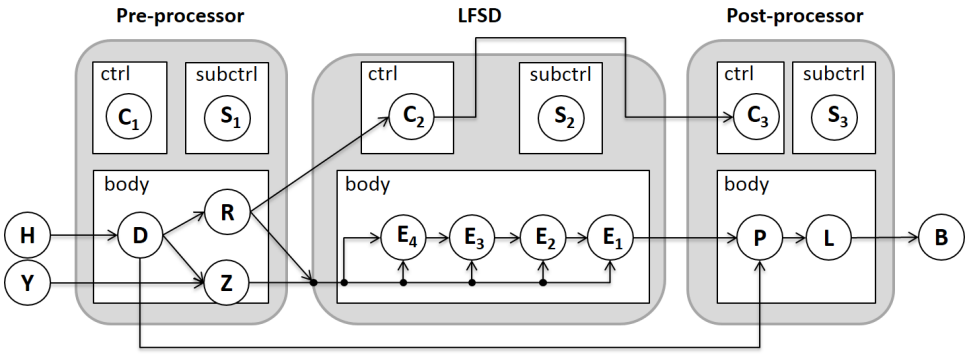

Figure 3: CF-PSDF model of soft MIMO detection application.

to reduce computational complexity. On the other hand, a large list is used for realizations in poor channel states to improve the detection accuracy. We consider three different settings of λ in our MIMO system implementation: $(1, 1, 1, P)$, $(1, 1, 2, P)$, and $(1, 2, 2, P)$. These settings result in $N_L = P$, $N_L = 2P$, and $N_L = 4P$, respectively.

In our CF-PSDF-based design, the `LFSD` actor includes three modes, `MODE-P`, `MODE-2P`, and `MODE-4P` to output $N_L = P$, $N_L = 2P$, and $N_L = 4P$ tokens (candidates), respectively. The associated control actor `C2` of the `LFSD` actor, when fired, computes the channel quality (instantaneous channel capacity of C, denoted ρ_C) with the input data exported from the `Pre-processor` actor, and then configures the subsystem mode parameter μ (i.e., selects a list size) based on the current channel quality indicator ρ_C. In the cases of $\rho_C > \rho_{TH1}$ ("good quality") and $\rho_C < \rho_{TH2}$ ("bad quality"), `MODE-P` and `MODE-4P` are selected, respectively, while in all other cases, the mode is set to `MODE-2P`. Here, ρ_{TH1} and ρ_{TH2} are two system parameters that determine the thresholds to use for determining good and bad channel quality, as described above.

The designer-provided specification of functional modes in CF-PSDF provides significant streamlining in the space of mode combinations that need to be handled during the implementation process. The body graph of the `LFSD` actor contains four actors — E_1, E_2, E_3, and E_4 — which, respectively represent the FSD processing elements for layers 1 through 4. Each E_i has four operational modes — a `LOAD` mode for reading input tokens, and three processing modes, denoted `M-1`, `M-2`, and `M-P`, to process data for $n_i = 1$, $n_i = 2$, and $n_i = P$, respectively. The total number of actor mode combinations in the body graph is therefore $4^4 = 256$. However, it is easy for the designer to understand and specify that only three of these combinations, which correspond to `MODE-P`, `MODE-2P`, and `MODE-4P` of the LFSD subsystem, are relevant. Thus, including the required control mode, the total number of operational modes for the LFSD subsystem is reduced from 256 to only 4 using the CF-PSDF convention of designer-specified functional modes.

4.2 Experimental Results

Our experiments on this MIMO detector case study are performed on a PC with an Intel 3GHz CPU and 4GB RAM. First, we compare the performance of the detector modeled in pure CFDF and CF-PSDF for a 4×4 MIMO system with QPSK, 16-QAM, or 64-QAM modulation. In the experiments, both implementations apply the canonical CFDF scheduler [5]; however, for the CF-PSDF-based implementation, the results of this scheduler are integrated with SDF schedules for individual functional modes, as described in Section 3.3.

Table 1: Performance comparison between CFDF- and CF-PSDF-based implementations. Run time is in microseconds.

Modulation	4-QAM		16-QAM		64-QAM	
Dataflow model	CF	CF-PS	CF	CF-PS	CF	CF-PS
Visited node count	272.6	68.7	1012	151.9	3972	484.4
Improvement	74.8%		85.0%		87.8%	
Run time	0.11	0.10	0.19	0.15	0.50	0.33
Gain	9.1%		21.1%		34.0%	

Table 1 lists experimental results for this comparison. From the results, we see that compared to CFDF, CF-PSDF modeling can significantly reduce the number of average visited actors per realization (shown in the row labeled *Visited node count*). Here, by a "visit", we mean a basic dataflow scheduling operation that involves assessing whether an actor has sufficient input data, firing the actor if it has sufficient data, or both. This reduction in visited node count, which can be viewed as a reduction in schedule execution overhead, arises because of the novel support in CF-PSDF for efficient quasi-static scheduling (i.e., in terms of local SDF schedules for individual functional modes). The overall performance of the CF-PSDF implementation is correspondingly improved as well. As we see from the row labeled "Run time", the average execution time is improved by 9.1%, 21.1%, and 34.0%, respectively, for QPSK, 16-QAM, and 64-QAM.

Next, we compare the performance of our dynamic MIMO detector against a conventional static detector with a fixed list size $N_L = 4P$ for a 64-QAM 4x4

Table 2: Experimental comparison between the SS and DS. Run time is in microseconds.

SNR (dB)	19.0	19.5	20.0	20.5	21.0
Run time (static)	0.425	0.425	0.424	0.426	0.425
Run time (dynamic)	0.370	0.353	0.339	0.327	0.318
Gain	12.9%	16.9%	20.0%	23.2%	25.2%

MIMO system (i.e., $P = 64$). To evaluate the coded BER performance, we feed the soft output of the detectors to a length 3600, rate 1/2 turbo decoder with eight iterations [13]. Our experimental results show that to achieve the target BER (assume 10^{-4}), the static system (SS) and dynamic systems (DS) require at least 19.84dB and 19.90dB signal to noise power ratio (SNR), respectively. In exchange for this small (0.06dB) degradation, the DS provides a significant improvement in run time (RT), as shown in Table 2. As expected, the RT of the DS at various SNRs is almost uniform. By contrast, the RT for the DS improves as SNR increases. This is because higher SNR provides more opportunities for use of smaller list sizes, which results in lower computational cost.

5 Conclusion

In this report, we have introduced a novel dataflow modeling approach, called CF-PSDF, that integrates core functional dataflow (CFDF) and parameterized synchronous dataflow (PSDF) techniques. CF-PSDF offers useful features including

flexible dynamic parameter reconfiguration and enhanced support for quasi-static scheduling. We have demonstrated the utility of CF-PSDF using a case study of soft MIMO detector implementation. Our experimental results show significant performance improvement through use of the streamlined scheduling techniques supported by CF-PSDF.

6 Acknowledgement

This research was sponsored in part by the Austrian Marshall Plan Foundation, and the US National Science Foundation (Grant No. CNS-1264486).

Bibliography

[1] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, Springer, 2010.

[2] S. Ha and E. A. Lee, "Compile-time scheduling and assignment of data-flow program graphs with data-dependent iteration," *IEEE Transactions on Computers*, vol. 40, no. 11, pp. 1225–1238, November 1991.

[3] B. Bhattacharya and S. S. Bhattacharyya, "Parameterized dataflow modeling for DSP systems," *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2408–2421, October 2001.

[4] J. McAllister, R. Woods, R. Walke, and D. Reilly, "Synthesis and high level optimisation of multidimensional dataflow actor networks on FPGA," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, 2004.

[5] W. Plishker, N. Sane, and S. S. Bhattacharyya, "A generalized scheduling approach for dynamic dataflow applications," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Nice, France, April 2009, pp. 111–116.

[6] L. Wang, C. Shen, and S. S. Bhattacharyya, "Parameterized core functional dataflow graphs and their application to design and implementation of wireless communication systems," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, Taipei, Taiwan, October 2013, To appear.

[7] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit, "Computation of buffer capacities for throughput constrained and data dependent inter-task communication," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, March 2008.

[8] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity — the Ptolemy approach," *Proceedings of the IEEE*, January 2003.

[9] B. Kienhuis and E. F. Deprettere, "Modeling stream-based applications using the SBF model of computation," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, September 2001, pp. 385–394.

[10] M. Wu, Y. Sun, G. Wang, and J. R. Cavallaro, "Implementation of a high throughput 3GPP turbo decoder on GPU," *Journal of Signal Processing Systems*, vol. 65, no. 2, 2011.

[11] E. Telatar, "Capacity of multi-antenna Gaussian channels," *European Transactions on Telecommunications*, vol. 10, no. 6, pp. 585–595, 1999.

[12] L. G. Barbero and J. S. Thompson, "Extending a fixed-complexity sphere decoder to obtain likelihood information for Turbo-MIMO systems," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 5, pp. 2804–2814, 2008.

[13] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," in *IEEE International Conference on Communications*, 1993, pp. 1064–1070.