April 16, 2009
Berkeley, CA

Ptolemy Miniconference

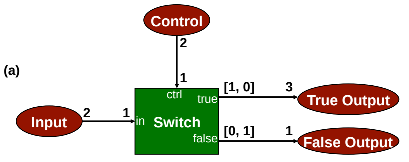
A Generalized Framework For Modeling And Scheduling Heterogeneous Dataflow Applications

Nimish Sane, William Plishker, Mary Kiemb, and Shuvra S. Bhattacharyya
Department of Electrical and Computer Engineering, University of Maryland, College Park

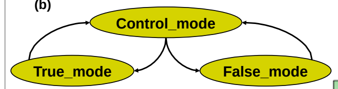
Introduction

- The growing complexity of modern signal processing applications along with their dynamic nature requires a generalized framework for efficiently modeling and scheduling such applications. We have developed the **core functional dataflow** (**CFDF**) model of computation that facilitates natural description of actors and provides a general framework for specifying static and dynamic dataflow applications. A variety of commonly used, existing dataflow models can be naturally represented in CFDF model thus allowing rapid prototyping for heterogeneous applications.
- We have extended our **MocGraph** package, a Java-based package supporting graph-theoretic analysis for models of computation (mocs) that has evolved from the graph package in Ptolemy II, to support trees and **generalized schedule trees** (**GST**s), in particular. A GST is a data structure for representing schedules of dataflow graphs employing a wide variety of dataflow models of computation.
- The CFDF model, along with the features of GST representations in MocGraph package, has enabled us to support simulations in **the DIF package** that can be used for verification and analysis of static and dynamic dataflow models.
- We have demonstrated the effectiveness of our recently developed scheduling techniques for heterogeneous dataflow applications modeled as CFDF graphs that exploit statically known mode transition patterns in dynamic dataflow actors.

(a)



(b)



(c)

Mode	No. of tokens consumed (produced) from (on) the ports			
	in	ctrl	true	false
Control_mode	0	1	0	0
True_mode	1	0	1	0
False_mode	1	0	0	1

(d)

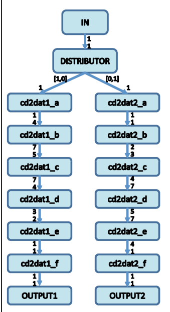
```
bool Switch.enable(mode)
    if (mode = Control_mode)
        then return (T(ctrl) ≥ 1)
    if (mode = True_mode or
            mode = False_mode)
        then return (T(in) ≥ 1)
```

```
Mode Switch.invoke(mode)
    if (mode = Control_mode)
        then if (token consumed from ctrl = true)
            then return True_mode
                else
                    return False_mode
    if (mode = True_mode)
        then consume 1 token from in and transfer it to true
            return Control_Mode
    if (mode = False_mode)
        then consume 1 token from in and transfer it to false
            return Control_Mode
```

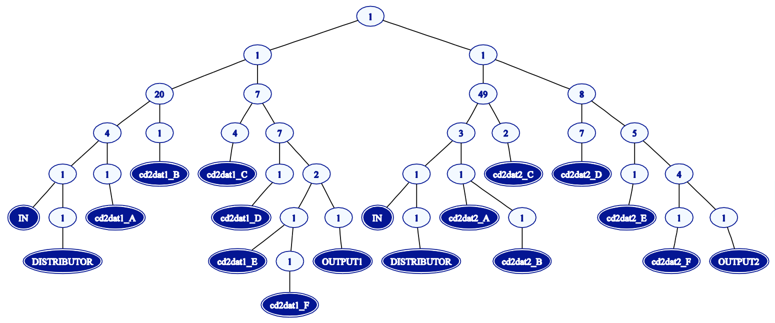
Fig. 1 The *Switch* actor: (a) a simple application; **(b)** transitions between the modes; **(c)** tokens consumed (produced) from (on) its inputs (outputs) in each mode; **(d)** the *enable* and *invoke* functions (T(*x*) denotes the number of tokens in the buffer connected to the port *x*).

Core Functional Dataflow

- Core functional dataflow is a deterministic form of enable-invoke dataflow (EIDF) [1]. EIDF is a generalized dataflow model which allows natural description of actors for static and dynamic dataflow models.
- Every actor is specified using a set of ***modes*** that capture the dataflow and functional behavior of the actor along with two methods ***enable*** and ***invoke***.
- The ***enable*** method checks if sufficient number of tokens are available on all of the actor inputs so that it can be fired in a given mode.
- The ***invoke*** method models the functionality of the actor. It is called only if the actor is *enabled* in that mode. It consumes specified number of tokens available on the actor inputs, produces required tokens on the outputs, and returns the next mode in which the actor must be fired.
- The dataflow behavior of an actor for a given mode is static in that number of tokens consumed (produced) by an actor from (on) its inputs (outputs) is fixed for any given mode of the actor.



(a)



(b)

Fig. (2) (a) Dual sample rate converter for converting sampling rates of two audio channels using a series of multirate FIR filters. Both the audio channels are interleaved and appear on the single input *IN*. The actor *DISTRIBUTOR* selects either of the sample rate converter depending upon the channel during the run-time. Each of the sample rate converters can be viewed as a multirate SDF application, while the *DISTRIBUTOR* introduces the dynamic behavior; **(b)** GST representation of the generalized CFDF schedule for the graph: internal nodes are annotated with schedule loop counts, while leaf nodes denote actors; double circle around the leaf node indicates a guarded execution.

Generalized Schedule Trees (GSTs)

- Ordered trees with **leaf nodes** representing **actors** and **internal nodes** representing **loop counts** for the schedule tree rooted at that node.
- Execution involves traversing the GST in depth-first manner and repeating a firing sequence for the number of times denoted by the loop count.
- **Unguarded execution** involves firing an actor every time the GST traversal algorithm reaches that node and hence, the scheduler must ensure that the actor is fireable.
- **Guarded execution** of an actor implies calling *invoke* method of the actor only if its *enable* method returns *true* for the given mode.
- GST representation is independent of the underlying dataflow model and hence can be used to represent schedules for a variety of known dataflow models. In particular, the CFDF model has allowed us to support functional simulations inside the DIF package (Functional DIF), based on the dataflow interchange format (DIF) language.

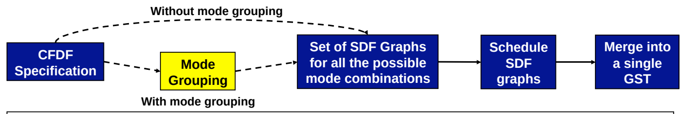


Fig. 3 Generalized scheduling strategy with and without mode grouping.

Generalized Scheduling Strategy

- We exploit the fact that for a given mode, each CFDF actor behaves like an SDF actor. We exploit the underlying static behavior of the CFDF actor to decompose the graph into multiple SDF graphs.
- These graphs are then scheduled using the known SDF scheduling strategies. These SDF schedules are then merged into a single schedule (as shown in Fig. 2(b)). The simulator switches between these schedules depending upon the current mode of the CFDF-actor.
- Though the number of SDF graphs can in general grow exponentially with the number of modes for each of the CFDF actors, in most of the practical applications, it is possible to eliminate most of the mode-combinations that can never occur and hence limit the number of SDF graphs to a much smaller number.
- It is also possible to explore the mode transition behavior. In most of the graphs it is possible to group certain modes, for which transitions have compile-time predictability e.g. transition from *True_mode* or *False_mode* to *Control_mode* for the *Switch* actor.
- This fact is then incorporated into the scheduling strategy before decomposing the graph into SDF graphs. This is more efficient in terms of buffer requirements and number of SDF components in the schedule, as shown in Table (2).

Application	Scheduling strategy	Simulation time (ms)	Maximum observed buffer sizes (no. of tokens)
Dual Sample Rate converter	Canonical	9148	9394
	Flat	1425	2408
	APGAN	1462	2234
Multi-PEA application	Canonical	2163	11198
	Flat	586	57
	APGAN	548	57

Table (1): Simulation times and buffer sizes for various scheduling strategies (without mode grouping) [3].

Application	Observed buffer-sizes (no. of tokens)		
	Without mode grouping	With mode grouping	Improve -ment (%)
B-spline interpolator*	479	304	37
Dual Sample Rate converter*	2278	2278	0
Multi-PEA application*	3802	2976	22

Table (2): Buffer requirements with and without mode grouping [4].
*To be discussed at the Design Automation Conference 2009 [4].

Conclusion and Future Work

- We have been able to model heterogeneous dataflow applications using CFDF model of computation. The portability of most of the existing dataflow models to CFDF allows leveraging existing scheduling and analysis techniques for these models to CFDF.
- CFDF based Functional DIF package has the feature of functional simulations allowing rapid prototyping of heterogeneous dataflow applications.
- We have developed a generalized scheduler and refined it further by introducing the concept of mode grouping. The effectiveness of both these schedulers has been demonstrated for various applications.
- We have provided support for GST in MocGraph package. The GST, we envision, will provide a standardized data structure for representing schedules independent of the underlying dataflow model.
- Though there have always been efforts to integrate test suits with the package, we would like to make testing a part of the development process and not as an afterthought. We plan to do this using a light-weight testing framework based on the DSPCAD instructional command-line environment (DICE) being developed at the DSPCAD research group.
- We continue to enhance the scheduling and analysis techniques for the CFDF semantics and add more application benchmarks into the DIF package.

References

1. W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya. Functional DIF for rapid prototyping. In *Proceedings of the International Symposium on Rapid System Prototyping*, pages 17-23, Monterey, California, June 2008.
2. W. Plishker, N. Sane, M. Kiemb, and S. S. Bhattacharyya. Heterogeneous System Design using Functional DIF. In *Proceedings of the International Symposium on Systems, Architectures, Modeling and Simulation*, pages 157-166, Samos, Greece, July 2008.
3. W. Plishker, N. Sane, and S. S. Bhattacharyya. Generalized Scheduling Approach for Dynamic Dataflow Applications. (To appear) In *Proceedings of the Design, Automation and Test in Europe Conference*, Nice, France, April 2009.
4. W. Plishker, N. Sane, and S. S. Bhattacharyya. Mode Grouping for More Effective Generalized Scheduling of Dynamic Dataflow Applications. (To appear) In *Proceedings of the Design Automation Conference*, San Francisco, California, July 2009.

UNIVERSITY OF MARYLAND