MAPPING MULTIMEDIA APPLICATIONS ONTO CONFIGURABLE HARDWARE WITH PARAMETERIZED CYCLO-STATIC DATAFLOW GRAPHS

Fiorella Haim*, Mainak Sen*, Dong-Ik Ko*, Shuvra S. Bhattacharyya*, and Wayne Wolf†
*Department of Electrical and Computer Engineering and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD, USA
†Department of Electrical Engineering, Princeton University, Princeton, NJ, USA

ABSTRACT

This paper develops methods for model-based design and implementation of image processing applications. We apply our previously developed meta-modeling technique of homogeneous parameterized dataflow (HPDF) [9] to the framework of cyclo-static dataflow (CSDF) [1], and demonstrate this integrated modeling methodology through hardware mapping of a gesture recognition application. We also provide a comparative study between HPDF/CSDF-based representation of the gesture recognition application, and a previously developed version based on applying HPDF in conjunction with conventional synchronous dataflow (SDF) semantics [9].

1. INTRODUCTION AND BACKGROUND

Real-time multimedia applications are widespread and modeling such applications effectively is critical for efficient implementation. Applying dataflow concepts in this context can provide valuable formal properties, such as efficient verification of bounded memory requirements and deadlock-free operation. Dataflow modeling also enables a wide variety of coarse-grain analysis techniques that can greatly improve memory requirements, performance estimation, and exploitation of parallelism [10]. Well-known dataflow modeling techniques, such as synchronous dataflow (SDF) [6] and cyclo-static dataflow (CSDF) [1] provide for especially strong analysis and verification; however, these are static models and cannot handle dynamically-structured applications, in particular, they cannot handle applications in which dataflow actors (functional modules) have dynamic data production and consumption rates.

Dataflow modeling techniques, such as Boolean-controlled dataflow [4] and its natural extension, integer-controlled dataflow [3], have been developed at the other extreme of the modeling trade-off between expressive power and analysis potential. These modeling techniques can support arbitrary applications (the models are Turing complete), but due to this generality, their support is weak for verification and optimization of implementations.

To provide useful new modeling alternatives in this space of trade-offs between expressive/analysis capability, a powerful meta-modeling technique called parameterized dataflow (PDF) was developed in [2] to handle dynamic applications. PDF is a hierarchical modeling approach. In PDF, dataflow graph parameters can be configured and reconfigured through associated *init* and *subinit* graphs of the application model, depending on the flexibility and relative frequencies with which parameters need to be reconfigured. Innovative techniques for analyzing PDF graphs and other forms of reconfigurable dataflow are developed in [8].

A related meta-modeling technique, called homogeneous parameterized (HPDF), was proposed in [9] to model data-dependent application structure with an emphasis on representing data transfers that are homogeneous across an edge. The homogeneity requirement in HPDF is in the sense that data transfer across an edge (production and consumption) must be equal (but not necessarily constant or statically-known) across corresponding invocations of the source and sink actors. While HPDF is targeted toward a somewhat narrower range of applications compared to PSDF, it is even more effective at enabling high-level optimization and verification when it is applicable. This is due to the provision for leveraging homogeneity in data transfer rates, while still allowing some degree of dynamic communication behavior. We discuss HPDF in more detail in the later sections.

Our work in this paper demonstrates the meta-modeling aspect of HPDF through our examination of alternative “base models” (cyclo-static and synchronous dataflow) to which HPDF can be applied. Furthermore, our work demonstrates significant benefits in memory management and performance efficiency that are attained through the careful mixing and matching of the orthogonal dataflow modeling methods that we apply. By exposing parallelism and data transfer patterns at a finer granularity in the model, memory requirements and associated energy consumption for memory accesses can be reduced. However, at the same time, we can still extract coarse-grain level parallelism effectively, and thus we can take advantage of pipelined architectures, and also we can retain the capability for coordinating execution through low overhead quasi-static schedules.

The remainder of this paper is organized as follows. Section 2 briefly reviews the HPDF meta-model, which was proposed in [9]. Section 3 develops the integration of HPDF and CSDF, which we denote as HPDF/CSDF. In addition to maintaining the simplicity and flexibility of the HPDF framework, HPDF/CSDF expresses more parallelism and helps to express applications at a finer granularity with respect to inter-actor data transfers. Section 4 describes a gesture recognition algorithm, and reviews a model for the application using HPDF concepts. Section 4 also analyzes the application dataflow model to optimize the memory organization. Section 5 then looks more deeply at the modeling of input to the application, as well as the dynamicity of the application, at a more detailed granularity. This analysis exposes application dataflow structure more thoroughly, yet in a manner that is concise enough to be used efficiently at design time. Section 6 explores scheduling strategies for the integrated HPDF/CSDF modeling methods developed in this paper. Section 7 presents experimental results that demonstrate more concretely the efficacy of our techniques. Section 8 presents a summary of the paper along with concluding remarks.

2. HPDF MODEL

2.1 HPDF modeling

In this section we briefly describe the HPDF modeling approach proposed earlier in [9]. HPDF is a meta-modeling technique; it can be applied to different dataflow models of computation (base models) to significantly enhance their expressive power.

2.2 HPDF Model Description

An HPDF subsystem is homogeneous in the sense that actors execute at the same average rate [9].

HPDF is a meta modeling technique. Hierarchical actors in an HPDF model can be refined using any dataflow modeling semantics that provide a well-defined notion of sub-system iterations. A hierarchical HPDF actor might have SDF, CSDF, PSDF.

or multidimensional SDF [7] actors as its constituent components.

As with many other dataflow models, such as SDF and CSDF, an HPDF edge e can have a non-negative integer delay $\delta(e)$ on it. This delay gives the number of initial data samples (tokens) on the edge.

Interface actors in HPDF can produce and consume arbitrary amounts of data, while the internal connections must, for fixed parameter values, obey the constraints imposed by the base model. An HPDF source actor in general has access to a variable number of tokens at its inputs, but obeys the semantics of the associated base model on its output. Similarly, an HPDF sink actor obeys the semantics of its base model at the input but can produce a variable number of tokens on its output.

Unlike PSDF, HPDF always executes in bounded memory whenever the component models execute in bounded memory.

3. INTEGRATING HPDF AND CSDF

A major contribution of this paper is demonstrating the integration of CSDF base model semantics into the HPDF meta-modeling framework. This integration provides simultaneous application of the bounded memory, dynamic parameterization of HPDF and the finer granularity, phased decomposition of actor execution in CSDF.

Recall that the homogeneity requirement in HPDF is in the sense that data transfer across an edge (production and consumption) must be equal (but not necessarily constant or statically-known) across corresponding invocations of the source and sink actors. In CSDF, a complete invocation of an actor involves execution of all of the phases in a fundamental period of the actor [1]. Integration of CSDF with HPDF allows the number of phases in a fundamental period to vary dynamically, and also allows the number of tokens produced or consumed in a given phase to vary dynamically. Such dynamic variation must adhere to the general HPDF constraint, however, that the total number of tokens produced by a source actor of a given edge in a given invocation (which, in the case of phased actors, means a given fundamental period) must equal the total number of tokens consumed by the sink in its corresponding invocation. Thus, for all positive n, the number of tokens produced by the nth complete invocation of a source actor must equal the number of tokens consumed by the nth complete invocation of the associated sink actor.

For fundamental periods that involve dynamic token transfer, this can be accommodated by employing a special token that delimits the end of a fundamental period of a source actor. The source actor produces this special end-of-invocation (EOI) delimiter just after the end of each complete invocation. The HPDF restriction then requires the following.

Suppose that the sink actor of a dynamically parameterized HPDF edge e consumes the last token in its ith invocation (fundamental period of phases) at time $z_i(t)$. Then just after completing $\delta(e)$ more consumption operations after time $z_i(t)$, the sink actor will consume an EOI token, and it will not consume any EOI tokens before that. This pattern must hold for all positive integers i (i.e., all invocation indices); that is, after each complete sink invocation, the next EOI token is consumed after exactly $\delta(e)$ consumption operations. Furthermore no EOI token should be consumed during the first invocation ($i = 1$) of the sink actor.

The above formulation is useful for precisely specifying how HPDF applies to dynamic parameterization of CSDF actors. The formulation can also be used to generate code for quasi-static schedules, and to verify consistency of HPDF specifications at run-time (i.e., to detect violations of HPDF behavior as soon as they occur).

4. A GESTURE RECOGNITION APPLICATION

In this section, we briefly describe the gesture recognition algorithm presented in [12], show how to model it with the previous HPDF model, and then show how to expose high-level application structure more effectively with the integration of HPDF and CSDF developed in Section 3.

4.1 Description of the algorithm

The gesture recognition algorithm is divided into a low-level processing algorithm that identifies human body parts in each frame, and a high-level processing algorithm that recognizes the gestures. We focus on the low-level processing algorithm, a block diagram of which is shown in Figure 1.

Region extraction classifies each pixel of a frame into skin-tone and non-skin-tone categories. Contour following determines the contours of the skin-tone regions. The Contour actor operates in two distinct modes — when it searches for a contour, it scans each row of the frame until it finds a starting point for a contour, then it goes to the other mode, where it follows the contour found before going back to searching for a new contour. Ellipse fitting finds ellipses to approximate the contours found previously. Graph matching classifies each ellipse as corresponding to a body part.

The diagram in Figure 1 expresses the application at a frame level. Here, the Region actor produces one token (one frame) for Contour to consume, and Contour outputs n contours that it finds in the input frame, out of which Ellipse successfully fits p ellipses and discards the rest. Match reads in all the p ellipses before it identifies body parts from the frame. The values n and p are input-dependent, and generally vary across actor invocations.

4.2 Fine granularity input modeling

The input to the gesture recognition system comes from a video camera. In our system, each frame of the video has 384 x 240 pixels and each pixel has 3 components, the luminance Y, and the chrominances Cr and Cb. We model the input as a cyclo-static actor having 384 x 240 = 92160 = (s) phases, where each phase corresponds to a different pixel. Using an adaptation of dataflow looped schedule notation (e.g., see [2]), the input (source) actor can be compactly represented as producing $(s\ 1)$ tokens in its fundamental period (1 token on each of s successive phases).

The static part of our system is now modeled as shown in Figure 2. The model captures now the pixel-level parallelism present in Region, and also expresses the frame regularity of the algorithm — i.e., after s phases, we start processing a new frame. With this regular, fine granularity CSDF representation, we can explore implementations with different architectures that may exploit both the pixel and frame parallelism.

In particular, we take advantage of this representation of the algorithm for our implementation in two aspects. First, we observe that the application can be pipelined into five blocks, each one processing a different frame. The second improvement is in the memory organization. Edges in a dataflow graph can be implemented as FIFO queues, but when dealing with images these buffers may

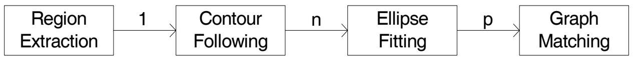

Figure 1. Block diagram of the gesture recognition algorithm.

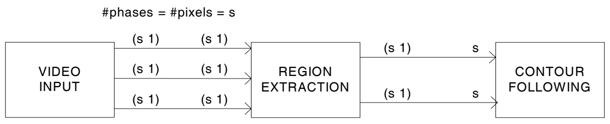

Figure 2. Model of the static part of the system

need to be larger than the internal memory available to them. In that case, external memory would be needed. Accessing external memory is power- and time-consuming, so when several banks are available, organizing the accesses carefully can reduce energy consumption significantly and improve the system performance in terms of throughput. Thus, the best memory organization scheme is the one that minimizes the number of accesses to it from an energy consumption point of view and minimizes the total number of non-overlapping accesses from a performance point of view.

We notice that for a sequential frame execution of the algorithm (i.e., without pipelining) the best possible memory organization for performance is to use all the possible memory bandwidth. This configuration should have minimum number of non-overlapping memory reads, thus optimizing the throughput. Interestingly, this configuration consumes the least energy as well.

If we implement a pipelined architecture, however, using the whole bandwidth is not the best option anymore. The memory management can be better adapted in this case if we swap banks so that an actor reads from a bank b data from frame i, while the preceding actor through edge e is writing in another bank data from frame $i+1$, where b and b' are "swapping banks" associated with edge e. In this way, an efficient memory organization consists of using half the banks available for each edge. An example with four memory banks is illustrated in Figure 3. In Section 7, we show some results obtained using this method with a multimedia FPGA board from Xilinx.

5. MODELING DYNAMICITY

In Figure 2, Contour needs to wait until the whole frame is available to start executing. Although this is true for the worst case, most of the times Contour can fire prior to having the whole frame. In order to model this, we divide the behavior of Contour into two phases, as shown in Figure 4: the first one scans the image looking for a contour and continues until it finds the starting point of one, thus consuming X_i pixels, without producing any output tokens; the second phase follows the contour and finds all the contours that are overlapping with each other. Now instead of processing the whole image, it will only process the subimage that goes from where a contour starts until all overlapping contours present are completed, thus consuming Y_i pixels. The output of this phase consists of k_i tokens. Each one of these output tokens is made up of a list of pixels belonging to a contour.

The HPDF condition as developed in Section 3 is respected here since no matter how the processing of Contour gets broken down into phases (based on the actual input image), the total number of pixels consumed in a frame by Contour equals the number of tokens produced for that frame by Region. That is,

$$s = \sum_i (X_i + Y_i)\,. \quad (1)$$

We model the input edge of Ellipse as a dynamically parameterized CSDF edge, capturing the dynamicity of Contour's production in the number of phases of Ellipse. The output of Ellipse is a set of parameters describing the ellipse whenever it can fit one to the contour. The Match actor has to wait to have all the p ellipses available before it can execute. It is easy to verify that the associated edges remain homogeneous in the sense of HPDF: an invocation of Contour will produce n tokens, based on the contours found in the current frame, while Ellipse will consume one token in each of its n phases (dynamically parameterized number of phases configured based on the pattern of EOI tokens on the edge). Similarly, Ellipse outputs tokens throughout n phases such that the sum of tokens over the phases is p, which is the number of tokens consumed by Match.

6. SCHEDULING

We can have different scheduling strategies depending on the implementation constraints that are most important; this is an advantage of dataflow modeling in general, and our development of HPDF enhances this advantage for the targeted class of applications. If data is passed between actors as vectors of different lengths, so that a static number of vector tokens whose lengths are dynamic are exchanged between a source and a sink, then we can have a very simple scheduler in place. In the HPDF/SDF model, if the edges $(\text{Contour}, \text{Ellipse})$ and $(\text{Ellipse}, \text{Match})$ receive and deliver one vector token each of length n and p respectively, then a valid schedule of the graph would be

$$VRCEM\,. \quad (2)$$

We can apply this concept of variable length tokens to find the schedule for a general HPDF graph. However if more granularity is expressed in the model, as we have done by integrating HPDF and CSDF, and if we want to exploit this finer granularity specification, we need to have a parameterized schedule, and to exchange data in a more fine-grained way. For our application, if it is modeled as in Figure 2 and Figure 4, a valid schedule would be:

$$(s\ V)(s\ R)(2I\ C)(n\ E)M\,. \quad (3)$$

In this schedule, the video input fires s times to provide the s pixels of a frame, while Region also fires s times, once per frame pixel. Contour fires $2I$ times, where I is the number of non-overlapping contours found in the current frame, since in its odd phases it searches for contours and in its even phases it follows the detected contours, and this happens I times in each frame. Ellipse fires n times, once for each contour and Match fires only once per frame. However, this basic schedule can be improved by grouping executions of Video and Region phases using the following modified schedule:

$$(s\ VR)(2I\ C)(n\ E)M\,. \quad (4)$$

Efficient quasi-static schedules of this form are enabled by the integrated HPDF/CSDF methodology that we have developed in this paper. Here, detection of EOI tokens, as described in Section 3, can be used to control the loops whose iteration counts are based on dynamically parameterized CSDF structures.

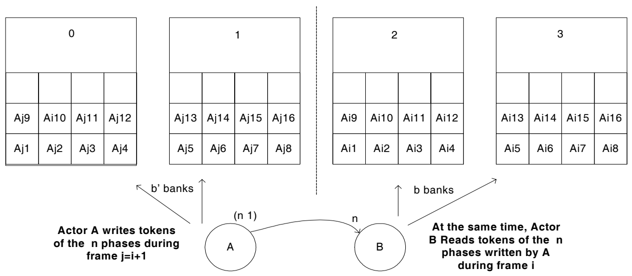
0
1
2
3
Aj9
Ai10
Aj11
Aj12
Aj1
Aj2
Aj3
Aj4
Aj13
Aj14
Aj15
Aj16
Aj5
Aj6
Aj7
Aj8
Ai9
Ai10
Ai11
Ai12
Ai1
Ai2
Ai3
Ai4
Ai13
Ai14
Ai15
Ai16
Ai5
Ai6
Ai7
Ai8
b' banks
b banks
(n 1)
n
A
B
Actor A writes tokens of the n phases during frame j=i+1
At the same time, Actor B Reads tokens of the n phases written by A during frame i

Figure 3. Example of swapping banks

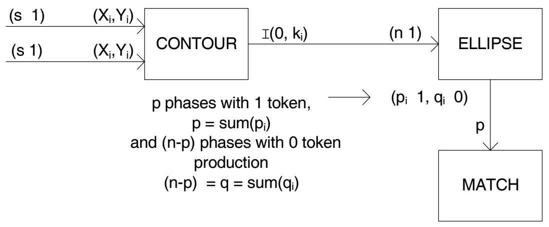

Figure 4. Model of the dynamic part of the system

7. EXPERIMENTAL RESULTS

For the gesture recognition application, we used the derived HPDF/CSDF model to determine an effective memory organization for our target platform, which is a *Xilinx Multimedia and Microblaze Development Board*. This board contains a Xilinx Virtex-2 field programmable gate array (FPGA) device. It also contains five integrated circuits that provide static RAM of size 512Kx32 bits with independent data, address, and control busses, and byte addressing capabilities. According to the memory datasheet, each bank consumes a maximum of 1.254 W when selected and 0.099 W otherwise. As per our implementation, write and read cycles take two clock periods. Assuming operation at the maximum memory frequency (133 MHz), we can calculate E, the energy needed to write and read the images as follows.

$$E = P \times (C/f_{max}) \times 2 , \qquad (5)$$

where P is the power consumed by the memory when selected, C is the total number of memory cycles and f_{max} is the maximum frequency. This analysis assumes that Contour only reads each memory location once, so internal buffers are available when needed.

In Table 1, we compare three cases with frames sized at 384x240 pixels, the last column shows the minimum memory size needed for each case for 5 independent banks with 32-bit words. The first case considers an effective memory organization that can be achieved from the information given by the HPDF/SDF model, considering a single frame. In this case, Region needs to read the frame from memory and then write its output to memory from where Contour will read it next. An appropriate configuration is achieved by storing the three components of the frame in the first 22.5K addresses of three memory banks and storing Region's output in 22.5K addresses of the other two memory banks. The second case is an efficient memory organization for a single frame that exploits the pixel-level parallelism shown with the precise modeling of the input stream (Figure 2). Region does not need to have a whole frame stored in memory anymore, and a more effective memory configuration is achieved when we write Region's output in the first 9216 addresses of the 5 banks.

Case 3 considers a stream of frames, extracting the frame level parallelism represented in the HPDF/CSDF model, where we swap between two pairs of banks to write Region's output in the first 23040 addresses in each case. In this case, at the same time that Contour is reading Region's output for frame i, Region is writing its output for frame $(i + 1)$.(Figure 3). Consequently, the buffer usage gets reduced from 184Kb between Region and Contour to 3 bytes while the other edges still have the same worst case buffer size as the previous representation [9]. This worst case arises when there is only one body part filling the whole image. However, in a typical scenario, the buffer sizes will be reduced significantly compared to the HPDF/SDF model of [9]

Table 1. Results of optimized memory organization in different scenarios.

Scenario	Total # of access cycles / frame	# of non-overlapping cycles / frame	Energy / frame (mJ)	Memory required
1 frame w/R	161280	69120	3.03	22.5 K
1 frame	92160	18432	1.73	9 K
Stream	92160	11520 (latency: 23040)	1.76	22.5 K

8. CONCLUSIONS

We have demonstrated the meta-modeling capabilities of HPDF, examining both SDF and CSDF as base models to which HPDF can be applied for alternative realizations of a gesture recognition application. Our HPDF/CSDF-based representation exposes pixel-level parallelism, enabling optimizations in buffer size and energy consumption. Furthermore, frame level parallelism can still be extracted from the model, providing information to generate an efficient schedule and explore pipelining options.

Useful directions for further work include hardware synthesis from HPDF/CSDF specifications; exploring the relationship to Compaan, which is a synthesis tool for MATLAB that uses internal representations having close relationships to dataflow graphs, and more recently, to restricted forms of dynamic dataflow [11]; integration into the dataflow interchange format (DIF), which provides a language, a tool development infrastructure, and software synthesis capabilities for working within and across different dataflow modeling techniques and design environments for DSP [5].

9. ACKNOWLEDGEMENT

This research was supported by grant number 0325119 from the U. S. National Science Foundation.

10. REFERENCES

[1] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete. Cyclo-static dataflow. *IEEE Transactions on Signal Processing*, 44(2):397-408, February 1996.

[2] B. Bhattacharya, S. S. Bhattacharyya. Parameterized Dataflow Modeling for DSP Systems. *IEEE Transactions on Signal Processing*. 49(10):2408-2410, October 2001.

[3] J. T. Buck. Static scheduling and code generation from dynamic dataflow graphs with integer-valued control systems. In *Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers*, pages 508-513, October 1994.

[4] J. T. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model*. PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, September 1993.

[5] C. Hsu, M. Ko, and S. S. Bhattacharyya. Software synthesis from the dataflow interchange format. In *Proceedings of the International Workshop on Software and Compilers for Embedded Systems*, Dallas, Texas, September 2005.

[6] E. A. Lee and D. G. Messerschmitt. Synchronous dataflow. *Proceedings of the IEEE*, 75(9):1235-1245, September 1987.

[7] P. K. Murthy and E. A. Lee. Multidimensional synchronous dataflow. *IEEE Transactions on Signal Processing*, 50(8):2064-2079, August 2002.

[8] S. Neuendorffer and E. Lee. Hierarchical reconfiguration of dataflow models. In *Proceedings of the International Conference on Formal Methods and Models for Codesign*, June 2004.

[9] M. Sen, S. S. Bhattacharyya, T. Lv, W. Wolf. Modeling Image Processing Systems with Homogeneous Parameterized Dataflow Graphs. *ICASSP 2005*, pp: V-133-V-136. March 2005.

[10] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., 2000.

[11] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprettere. System design using Kahn process networks: the Compaan/Laura approach. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, February 2004.

[12] W. Wolf, B. Ozer, T. Lv. Smart cameras as embedded systems. *IEEE Computer Magazine* Vol 35, Iss 9, Sept 2002, pg48-53