MODELING IMAGE PROCESSING SYSTEMS WITH HOMOGENEOUS PARAMETERIZED DATAFLOW GRAPHS

Mainak Sen[1], Shuvra S. Bhattacharyya[1], Tiehan Lv[2], Wayne Wolf[2]

[1]{mainak,ssb}@eng.umd.edu
Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies,
University of Maryland, College Park, MD, 20742, USA.
[2]lv@ee.princeton.edu, wolf@princeton.edu
Department of Electrical Engineering,
Princeton University, Princeton, NJ, 08544, USA.

ABSTRACT

In this paper, we describe a new dataflow model called *homogeneous parameterized dataflow* (*HPDF*). This form of dynamic dataflow graph takes advantage of the fact that in a large number of image processing applications, data production and consumption rates, though dynamic, are equal across graph edges for any particular iteration, which leads to a homogeneous rate of actor execution even though data production and consumption values are dynamic and vary across graph edges. We discuss existing dataflow models and formulate in detail the HPDF model. We develop examples of applications that are described naturally in terms of HPDF semantics and present experimental results that demonstrate the efficacy of the HPDF approach.

1. PREVIOUS WORK

Real-time multimedia applications are an integral part of embedded systems technology. Modeling such applications using dataflow graphs can lead to useful formal properties, such as bounded memory requirements, and efficient synthesis solutions (e.g, see [2]). The synchronous dataflow (SDF) model for example has particularly strong compile time predictability properties [6]. However, this model is highly restrictive and cannot handle data-dependent execution of dataflow graph vertices (*actors*). There have been previous studies on extensions of SDF to provide for more flexible actor execution, including handling of such dynamic execution capabilities. For example, a cyclo-static dataflow (CSDF) [3] graph can accommodate multiphase actors with different consumption and production rates at the input and output, respectively, at different phases of iteration. This provides for more flexibility but does not permit data dependent production or consumption patterns. Another extension known as the token flow model [4] was proposed in which we can have dynamic actors where the number of data values (*tokens*) transferred across a graph edge may depend on the run-time value of a token that is received at a "control port" of an incident actor. A meta-modeling technique called parameterized dataflow [1] (PSDF) was proposed later in which dynamic dataflow capability was formulated in terms of run-time reconfiguration of actor and edge parameters. We elaborate further on the PSDF model in the following section.

2. PARAMETERIZED DATAFLOW MODELING

We discuss in brief the parameterized dataflow model proposed in [1]. Parameterized dataflow is a meta-modeling technique. It can be applied to any underlying base dataflow model that has a well-defined notion of a *graph iteration*. The model increases the expressive power of the base model by providing for run-time reconfigurability of actor and edge parameters in a certain way. When parameterized dataflow is applied to SDF as the base model, the resulting parameterized synchronous dataflow (PSDF) model can be viewed as an augmentation of SDF that incorporates run-time reconfiguration of parameter configurations for actors and edges.

An actor A in PSDF is characterized by a set of parameters ($params(A)$) that control the actor's functionality and dataflow behavior. Each parameter is either assigned a value from a set of viable values or left unspecified. These unspecified parameters are assigned values at run-time through a disciplined run-time reconfiguration mechanism. Techniques have been developed to execute PSDF graphs efficiently through carefully constructed quasi-static schedules (schedules for dynamic dataflow graphs in which a significant portion of sequencing decisions are fixed at compile time) [1].

PSDF specifications are built up in a modular way in terms of hierarchical subsystems. Every subsystem is in general composed of three subgraphs, called the *init*, *subinit* and *body* graphs. New parameter values to use during run-time reconfiguration are generally computed in the init and subinit graphs, and the values are propagated to the body graph, which represents the computational core of the associated PSDF subsystem. The init graph is invoked at the beginning of each invocation of the (hierarchical) parent graph and the subinit graph is invoked at the beginning of each invocation of the associated subsystem followed by the body graph.

3. GENERIC MODEL FOR HIERARCHICAL RECONFIGURATION OF DATAFLOW GRAPHS

Parameterization is a widely-used method to implement dynamic behavior of a dataflow graph. But a parameterized actor might also have a predetermined production and consumption rate. For example, an FIR filter might have its number of taps as a parameter, which does not affect the production consumption rate. In this paper, we discuss parameters in the context of actors whose token production and consumption rates are a function of these parameters. In [9], the authors develop a mathematical model to represent the reconfiguration of various types of dynamic dataflow graphs. The model allows reconfiguration at all levels of hierarchy. A hierarchical reconfiguration model is represented by a *containment tree,* which has a finite set of actors in it. Non-leaf nodes are composite actors and leaf elements are atomic actors. The behavior of a composite actor is given by the actors that are its direct children. Every actor has its own set of parameters which define its behavior and there is a one-to-one relation between the parameters and actors. Dependencies among parameters are expressed explicitly through a domain function and its value is constrained by a *constraint function*. A dependent parameter must at all times satisfy the constraint function to become *consistent*. An independent parameter has null in its domain function.

The authors introduce specific points in their model called *quiescent points*, which are constrained points in the execu-

tion model where change of parameter values are permitted. These points occur between firings and an actor cannot communicate or perform computation at these points. A *precedence relation* is set that performs partial ordering of quiescent points of all the actors. At each quiescent point, a set of independent parameters Q is chosen for reconfiguration and all the parameters dependent on Q are also reconfigured based on their initial and reconfigured values. Parameters that cannot be reconfigured or can be changed only at certain quiescent points are declared as *constant parameters*. A constant parameter can be forced to remain constant either during one particular execution of the model or over firings of the associated actor. To statically analyze the reconfiguration of a model, two methodologies have been suggested. Firstly, all the executions of the model are checked along with all possible reconfigurations and any invalid reconfiguration predicts invalidity of the model. Secondly, the authors suggest a least change context for every parameter p which is a conservative estimate of the actors affected by p. This helps in easy semantic constraint checking.

4. HOMOGENEOUS PARAMETERIZED DATAFLOW MODEL (HPDF)

In this section we develop the HPDF model, which like parameterized dataflow is a meta-modeling technique in that it can be applied to different dataflow base models. We present the characteristics of the actors, edges and delay buffers in an HPDF graph.

4.1 HPDF Model Definition

An HPDF subsystem is homogeneous in two ways. First, unlike general SDF graphs and other multirate models, the top level actors in an HPDF subsystem execute at the same rate. Second, unlike the hierarchical parameterized dataflow semantics, reconfiguration across subsystems can be achieved without introducing hierarchy (i.e., across actors that are at the same level of the modeling hierarchy) when it is more natural to do so. At the same time, hierarchy can be used when desired.

HPDF is a meta modeling technique. Composite actors in an HPDF model can be refined using any dataflow modeling semantics that provides a well-defined notion of sub-system iteration. So the composite HPDF actor might have SDF, CSDF, PSDF or MDSDF [7] actors as its constituent actors.

HPDF edges can have non-unity delays (initial tokens) on them as in other dataflow models. Furthermore, the stream of tokens that is passed across an edge needs markers of some kind to indicate the "packets" that correspond to each iteration of the producing/consuming actors. An end-of-packet marker is used for this purpose in our implementation.

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 it 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.

4.2 Comparison of HPDF and PSDF

As we mentioned in Section 4.1, in HPDF we do not have to introduce hierarchy in the form of subsystems to account for dynamic behavior of actors. Suppose a dynamic source actor A produces n tokens that are consumed by the dynamic sink actor B. In PSDF, we need to have A and B in different subsystems; the body of A would set the parameter n which will be a known quantity at that time, in subinit of B. This hierarchy can be avoided in HPDF as we assume that data is produced and consumed in same-sized blocks. As we will describe further in Section 5, this simple form of dynamicity has many applications in signal, image and video processing algorithms. It therefore deserves explicit, efficient support as provided by HPDF.

5. APPLICATIONS

In this section, we discuss interesting applications of HPDF in real-time video analysis for gesture recognition and face detection.

5.1 A Gesture Recognition Algorithm

Figure 1 shows a part of the algorithm flow to identify body parts and categorize their movements in a succession of images as described in [11]. 'Region finding' eliminates the background and separates skin-tone and non-skin tone foregrounds from the input image file. 'Contour following' generates a contour for each of the regions in the foreground based on the regions found in the previous step. 'Ellipse fitting' fits an ellipse to each of the contours. 'Graph matching' identifies various body parts by comparing the group of ellipses with a library of graphs.

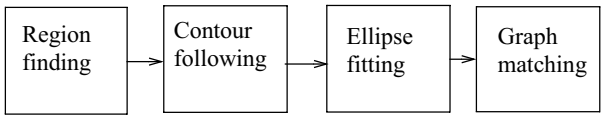

Figure 1. Top level representation of the video analysis algorithm.

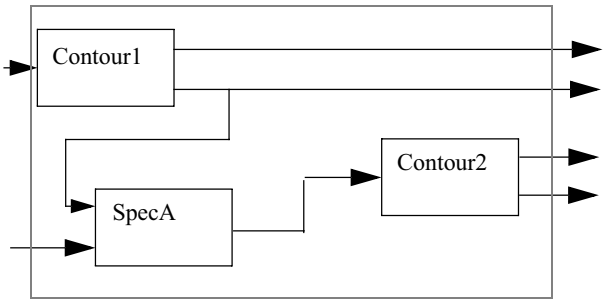

Figure 2. 'Contour following' is composed of these actors. Contour1 and Contur2 take one image as input and output N regions. SpecA is a specifier.

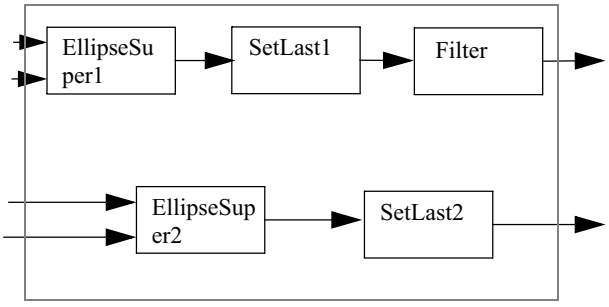

Figure 3. 'Ellipse fitting' is composed of the actors shown above. Filter takes one token as input and outputs zero or one token.

The above mentioned algorithm is dynamic as far as the production and consumption of data among various blocks is concerned. To analyze the dynamicity, we concentrate on the blocks

that input and output variable amounts of data that can only be determined during runtime. The 'region finding' block takes two images as input and outputs a background subtracted skin-toned image. Therefore the input and output production/consumption rates of this block are static. 'Contour following' takes the skin-toned image as input and outputs N regions corresponding to the number of contours it can find. N is understandably determined during run-time. 'Ellipse fitting' has a filter that takes one token as input. The token contains an array of numbers. Zero or one tokens are output depending on whether the first element of the input array is zero. 'Graph matching' takes all the ellipses that were fitted on the contours before it looks for a match in the library of graphs.

Thus, the edges in Figure 1 between 'Contour following' and 'Ellipse fitting' and between 'Ellipse fitting' and 'Graph matching' have variable amounts of data that depend on the input image. Also 'Graph Matching' has to wait until all the fitted ellipses are available before it can execute. The HPDF model is well suited for this kind of dataflow. The idea is that each of the modules in Figure 1 can be made to execute at the same rate producing correct output, so they are homogeneous in this sense. To ensure homogeneity in firings, we effectively wrap the variable number of tokens produced in the dynamic actors in one token and use it as a variable-size vector token in our implementation.

5.2 HPDF Model for the Application

We implemented the HPDF model along with the gesture recognition system in the Ptolemy II framework [5], which is a popular design tool for experimenting with models of computation in embedded system design. The application in the top level representation of our implementation has four composite (hierarchical) actors, namely Region, Contour, Ellipse and Match. Each composite actor has its own "director," which is the software module in Ptolemy II that effectively implements the associated model of computation, which in this case is an SDF director for all four of the composite actors. In general, each composite actor can have a different director (SDF, CSDF, MDSDF, etc.).

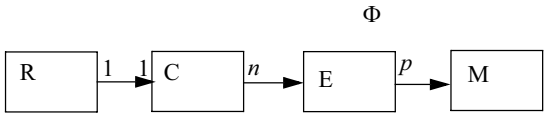

Figure 4. HPDF model of the application with parameterized token production and consumption rate of C and E shown with n and p, respectively.

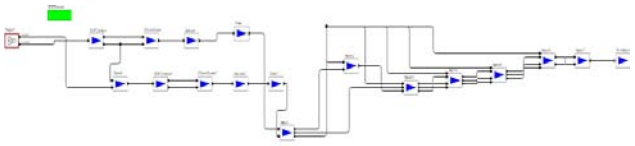

Figure 5. The HPDF graph of the above application with flattened hierarchy for C, E and M in Ptolemy II.

In Figure 4, R, C, E, M represent Region, Contour, Ellipse, and Match, respectively. Figure 5 shows the HPDF graph of the image processing algorithm with the hierarchy flattened. As already mentioned, the contour module produces a dynamic amount of data because it outputs the number of contours it finds in the input image, which is image-dependent and therefore cannot be determined statically. For the input images we applied in our experiments, the contour actor produces three regions.

Now consider the scheduling process for the system in Figure 4. Denote by n the number of regions output by the contour actor (again, this is a quantity that may vary from iteration to iteration). Ellipse consumes the n regions and produces p ellipses that fit them. Match requires all the p ellipses before it can execute. All the other edges produce and consume one data unit per firing. We pass the dynamic amount of data as a single vector token, so C produces one vector token of length n and E produces one vector token of length p. So the schedule of the graph Φ would be $RCEM$. However, to expose more parallelism, it is efficient to decompose E into the refined subsystem shown in Figure 6. Here, E' represents an actor which tries to fit an ellipse to each contour it receives at its input - thus outputs one token (ellipse in our case) on success and zero token on failure.

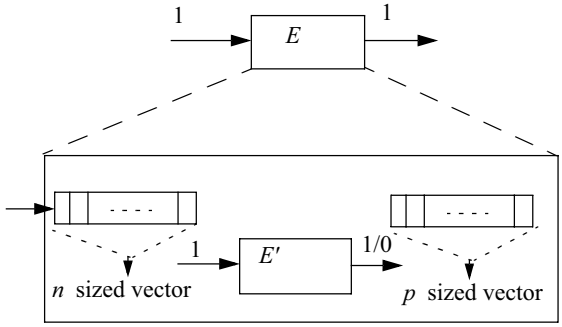

Figure 6. Inside of actor E. E' consumes one token per invocation and produces either one or zero tokens.

Taking all of this into consideration, a valid quasi-static schedule, which is similar to the form of a parameterized looped schedule [1], for Φ is $RC(nE')M$, where the parenthesized term (nE') represents a loop that repeats n iterations of E'.

5.3 A Face Detection Algorithm

We also modeled an image-based face detection algorithm [8] summarized using HPDF to further demonstrate the efficacy of the HPDF modeling approach.

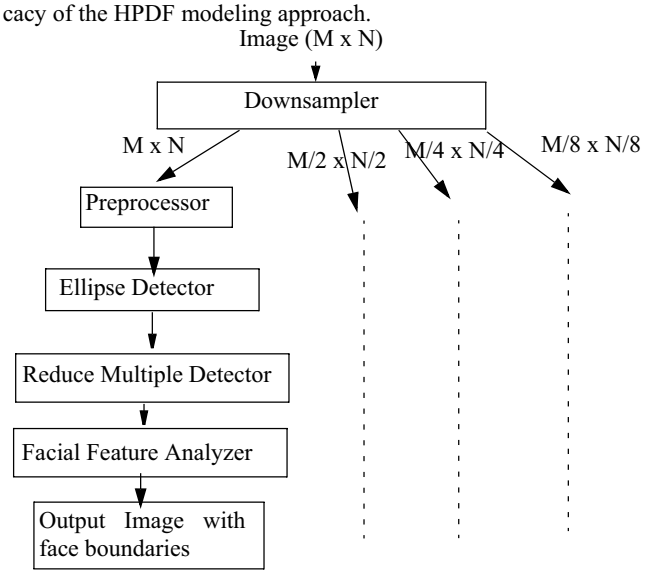

Figure 7. A face detection algorithm. Processing with downsampled images not shown explicitly.

The algorithm assumes that the facial feature analyzers are trained with an extensive set of data for recognizing features in faces. The downsampler takes care of zoomed-in faces. For each zoomed-in image, we perform zero mean and contrast enhancement — collectively shown as preprocessing in Figure 7. In 'Ellipse Detector', we break the image into windows of the size of faces that were used to train the classifiers and look for ellipses. This breaking down simulates the sequential sliding window in the parallel domain. The number of ellipses detected is dynamic as it depends on the input image. In the 'Reduce Multiple Detector', we remove some of the ellipses with close features to some existing ellipses, suggesting that the same ellipses were detected multiple times. All the ellipses can be given to the 'facial feature analyzer', which outputs a binary value of 1 or 0 if the trained analyzer detects a face or not, respectively. The output of the algorithm is the original image with face boundaries marked with rectangles.

5.4 HPDF model for face detection algorithm

We modeled the application shown in Figure 7 in HPDF. The resulting model represents the face detection algorithm for one particular scaling of the input image in HPDF as shown in Figure 8. *DS*, *P*, *ED*, *RMD*, *FFA* represent the downsampler, preprocessor, ellipse detector, reduce multiple detector and facial feature analyzer, respectively. The output of *DS* is a downsampled image of size *mXn* where $m = M/2^i$, $n = N/2^i$, i is any integer and M, N are the length and width of the original image in pixels. *ED* is parameterized since the number of outputs from it depend on the input image. The output of *ED* is a vector of size q where q is number of ellipses the algorithm finds in the input image. The output of *RMD* is a vector of length p where each entry signifies a unique ellipse detected. *FFA* outputs a vector of length r with all the locations of detected faces in the input image.

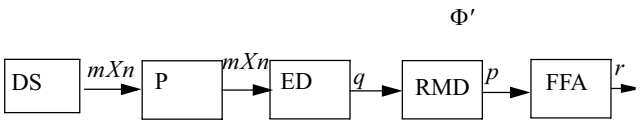

Figure 8. HPDF model for the face detection algorithm mentioned in section 5.3

6. IMPLEMENTATION RESULTS

We developed a Texas Instruments (TI) programmable digital signal processor implementation of the HPDF model of the gesture recognition algorithm described in section 5.1. We evaluated this implementation on TI Code Composer Studio version 2 for the C'6xxx family of programmable DSP processors. The application when implemented with our HPDF model for a C64xx fixed point DSP processor has a runtime of 21405671 cycles and with a clock period of 40 ns, the execution time was calculated to be 0.86 sec. The scheduling overhead for the implementation is minimal as the HPDF representation inherently leads to a highly streamlined quasi-static schedule as described in Section 5.2. The worst case buffer size for an image of 348 X 240 pixels was 184 kilobytes on the edge between region and contour, 642 Kb between contour and ellipse and 34 Kb between ellipse and match for at total of 860 kilobytes. The original code (without modeling) had a runtime of 27741882 cycles and with the same clock period of 40ns, the execution time was 1.11 sec. So our model improved the execution time by 23 percent.

7. CONCLUSION

This paper has developed HPDF, an efficient meta-modeling technique for capturing a commonly-occurring, restricted form of dynamic dataflow relevant to image processing applications. Applications that exhibit data-dependent token traffic and have an aggregating actor in the later stage of the algorithm to combine results appear to fit the HPDF paradigm particularly well. For example, in a number of image processing applications, the image is divided into smaller subimages for faster implementation, each subimage leads to a data-dependent volume of data production, and the produced data from all of the subimages is combined later to get the final result. HPDF captures the inherent dataflow structure in such applications without going into more complicated hierarchical representations or into more general dynamic dataflow modeling approaches where key analysis and synthesis problems become impossible to solve exactly. Also HPDF models can be synthesized into efficient software implementations through low overhead quasi-static schedules. Useful directions for future work include exploring the automated derivation of efficient, synthesizable hardware implementations from HPDF representations, for example, using the hardware synthesis framework developed earlier [10].

8. ACKNOWLEDGEMENTS

This research was supported by grant number 0325119 from the U. S. National Science Foundation. The authors are thankful to Vikas Raykar and Gaurav Aggarwal for helping out with the face detection algorithm.

9. REFERENCES

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

[2] S. S. Bhattacharyya, R. Leupers, and P. Marwedel. Software synthesis and code generation for DSP. *IEEE Transactions on Circuits and Systems — II: Analog and Digital Signal Processing*, 47(9):849-875, September 2000.

[3] G Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cyclo-Static Dataflow. *IEEE Transactions on Signal Processing*. Vol 44, No 2, February 1996.

[4] J. T. Buck. A Dynamic Dataflow Model Suitable for Efficient Mixed Hardware and Software Implementations of DSP Applications. *Proceedings of the 3rd international workshop on Hardware/software co-design.* Pages 165-172, 1994.

[5] 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.

[6] E. Lee and D. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, pages 55–64, September 1987

[7] E. A. Lee. Multidimensional Streams Rooted in Dataflow. *Proceedings of the IFIP Working Conference on Architectures and Compilation Techniques for Fine and Medium Grain Parallelism,* January 1993.

[8] H. Moon, R. Chellappa, A. Rosenfeld. Optimal Edge-based Shape Detection. *IEEE Transaction on Image Processing,* Vol 11(11), pp 1209-1226, 2002.

[9] S. Neuendorffer, E. Lee. Hierarchical Reconfiguration of Dataflow Models. *Conference on Formal Methods and Models for Codesign (MEMOCODE)*, June 22-25, 2004.

[10] M. Sen, S. S. Bhattacharyya. Systematic Exploitation of Data Parallelism in Hardware Synthesis of DSP Applications. *ICASSP 2004.*

[11] W. Wolf, B. Ozer, T. Lv. Smart cameras as embedded systems. *IEEE Computer Magazine* Vol 35, Iss 9, Sept 2002, Pages 48-53.