MULTIDIMENSIONAL DATAFLOW GRAPH MODELING AND MAPPING FOR EFFICIENT GPU IMPLEMENTATION

Lai-Huei Wang[1]*, Chung-Ching Shen*[1]*, Gunasekaran Seetharaman*[2]*, Kannappan Palaniappan*[3]*, and Shuvra S. Bhattacharyya*[1]
[1]Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742, USA
{laihuei, ccshen, ssb}@umd.edu
[2]Air Force Research Laboratory, Rome, NY, USA
Gunasekaran.Seetharaman@rl.af.mil
[3]University of Missouri-Columbia
palaniappank@missouri.edu

ABSTRACT

Multidimensional synchronous dataflow (MDSDF) provides an effective model of computation for a variety of multidimensional DSP systems that have static dataflow structures. In this paper, we develop new methods for optimized implementation of MDSDF graphs on embedded platforms that employ multiple levels of parallelism to enhance performance at different levels of granularity. Our approach allows designers to systematically represent and transform multi-level parallelism specifications from a common, MDSDF-based application level model. We demonstrate our methods with a case study of image histogram implementation on a graphics processing unit (GPU). Experimental results from this study show that our approach can be used to derive fast GPU implementations, and enhance trade-off analysis during design space exploration.

Index Terms— Dataflow graph, multidimensional synchronous dataflow, graphics processing unit, integral histogram.

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]). Dataflow provides a formal mechanism for describing specifications of DSP applications, imposes minimal data-dependency constraints in specifications, and is effective in exposing and exploiting task or data level parallelism for achieving high performance implementations. Synchronous dataflow [2] has been popular in design of DSP applications because of its useful features, including compile-time, formal validation of deadlock-free operation and bounded buffer memory requirements, as well as support for efficient scheduling and buffer size optimization [1]. However, the SDF model is well suited only for one-dimensional DSP algorithms, such as those in the domains of speech, audio, and digital communication. Multidimensional synchronous dataflow (MDSDF) [3] is a generalization of SDF to multiple dimensions. MDSDF provides an effective model for a variety of multidimensional DSP systems that have statically structured dataflow characteristics.

In this paper, we develop new methods for efficient implementation of parallel processing solutions for signal processing systems using MDSDF representations. Our proposed design methods apply dataflow transformations to exploit data parallelism hierarchically for multidimensional dataflow graphs. Our design methods provide a systematic approach for exposing and exploiting parallelism from multidimensional dataflow specifications across different levels of the specification hierarchy. We demonstrate our proposed new modeling techniques and design methods by applying them to optimize implementations on the NVIDIA graphics programming unit (GPU) programming model [4]. Using our new MDSDF-based design techniques, we demonstrate efficient GPU implementations for integral histogram computations, which form an important class of image processing operations for surveillance and monitoring applications. The results of our experiments demonstrate concretely that our proposed design methods are effective in mapping formal design models for multidimensional DSP systems into efficient implementations on complex multicore processors.

2. RELATED WORK

A variety of dataflow based design tools has evolved in recent years for design and implementation of signal processing systems (e.g., see [1, 5, 6]). In this section, we summarize a number of recent efforts beyond MDSDF (see Section 1) that have

978-0-7695-4856-2/12 $26.00 © 2012 IEEE
DOI 10.1109/SiPS.2012.10

focused especially on multidimensional dataflow modeling.

Keinert et al. propose an extension of MDSDF, called windowed synchronous dataflow (WSDF) [7]. WSDF allows modeling of sliding window algorithms for a multidimensional applications. Array–OL [8] is a language devoted to applications that involve multidimensional intensive signal processing. Two levels of description are used for modeling parallelism in Array–OL — one is the global model for defining task parallelism, while the other is the local model for expressing data parallelism. Blocked dataflow (BLDF) [9] provides meta-modeling semantics that can be used to represent block-based and multidimensional processing in terms of different specialized dataflow models. BLDF provides a unified framework that leads to efficient dataflow graph scheduling and memory management.

McAllister et al. [10] augment the MDSDF model with parameterized array expressions. Their modeling approach, called Multidimensional Arrayed Synchronous Dataflow (MASD), provides graph range parameters to control token dimensions at input and output ports, which enables systematic trade-off exploration between actor network size and token size.

The distinguishing contribution of this paper is that it presents a novel design method, building on the MDSDF model of computation, for hierarchical exploitation of parallelism in DSP applications. The method developed in this paper helps to expose multidimensional parallelism at different design levels in a platform-independent way, and to exploit such parallelism using suitable platform-specific mapping optimizations at the back-end of the design flow. Graph clustering and MDSDF dataflow analysis are applied in novel ways to provide a systematic approach for mapping applications to DSP platforms that employ parallelism at multiple levels. In addition to motivating and concretely illustrating our proposed design method, we demonstrate its utility through a case study of an important, practical multidimensional signal processing application.

3. MODELING

In this section, we present a structured design method based on MDSDF graphs for hierarchical mapping of DSP systems onto parallel architectures. In various forms of data parallel programming, programmers can define functions, and have multiple calls to the functions execute in parallel on different data sets (e.g., see [4, 11]). Recent data parallel programming environments emphasize support for exploiting multi-level or *hierarchical* parallelism, where parallelism is exploited programmatically at multiple levels of granularity. For example, CUDA [4] provides a two-level thread hierarchy, where a set of threads makes up a *thread block*, and multiple thread blocks form a *grid*.

Such hierarchical support for representing parallelism is important for multidimensional signal processing applications, where parallelism exists in different forms at different levels of the *design hierarchy* (*DH*) (e.g., inter-frame, inter-block, and inter-pixel parallelism in video processing). In this section, we build on the MDSDF model of computation, and develop a design method to represent and apply parallelism hierarchically for multidimensional dataflow graphs.

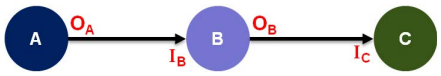


Fig. 1. An example of a three-actor MDSDF graph.

Let $G = (V, E)$ denote an MDSDF graph where $V = \{v_1, v_2, ..., v_L\}$ is a set of vertices (*actors*), and $E = \{e_1, e_2, ..., e_K\}$ is a set of directed edges, which represent communication between actors according to MDSDF semantics. In MDSDF graphs, actor firings are indexed (in their associated "firing spaces") by n-dimensional vectors, where the values of n depend on the dimensions of the data that are produced and consumed ($n = 1$ corresponds to conventional single-dimensional, SDF-like firing sequences) [3].

Suppose that v is an MDSDF actor with a firing space of M dimensions, and let $r_{v,i}$, for $i = 1, 2, \ldots, M$, denote the size of the ith dimension of the firing space for v in a given periodic schedule S for G. A periodic schedule is a sequence of actor firings that executes each actor at least once and produces no net change in the numbers of tokens queued on the edges of G [2, 3]. We refer to the M-vector $r_v = [r_{v,1}, r_{v,2}, ..., r_{v,M}]$ as the *firing vector* for actor v associated with S. The product of the M elements of this vector gives the total number of firings of v within S. For a properly constructed MDSDF graph, r_v can be computed by solving a system of equations called the *balance equations* for the graph [3].

Consider, for example, the 3-node graph illustrated in Fig. 3. The firing vectors r_A, r_B, and r_C can be found by solving the following balance equations for $i = 1, 2, ..., M$:

$$r_{A,i}O_{A,i} = r_{B,i}I_{B,i}, \quad r_{B,i}O_{B,i} = r_{C,i}I_{C,i}, \qquad (1)$$

where $I_X = [I_{X,1}, I_{X,2}, ..., I_{X,M}]$ and $O_X = [O_{X,1}, O_{X,2}, ..., O_{X,M}]$ are the M-dimensional consumption and production rates, respectively, for actor X.

Now suppose that we have an N-level hierarchical parallel programming model (*platform hierarchy*) P, which we want to use to implement a given MDSDF graph G. For example, such a parallel programming model could be used as a target for code generation or could be used for an implementation that is derived from hand based on a functional reference ("golden model") that is based on the MDSDF specification. We develop an N-level hierarchical dataflow graph transformation approach to achieve such a mapping from MDSDF to P. We refer to N in this context as the *platform depth*.

First, we introduce some definitions and notation related to hierarchical dataflow graphs. For a dataflow graph $G =$

(V, E), let $P_i(V)$ and $P_o(V)$ be the sets of input and output ports of all actors in V, respectively. A *supernode* s in G is an actor (i.e., $s \in V$) that is associated with a "nested dataflow graph" $H(s)$, where execution of s in G corresponds to execution of $H(s)$. In general, not all actor ports in $H(s)$ are connected in $H(s)$ (i.e., not all of them connect to edges within $H(s)$). The "unconnected actor ports" are referred to as the *interface ports* of $H(s)$, and these ports are in one-to-one correspondence with ports of actor s.

If G is the "top" of the DH (i.e., G is not encapsulated by a supernode in another graph), then we say that the *nesting level* (or simply *level*) of G, denoted $\lambda(G)$, is 1. Similarly, for each supernode s in G, $\lambda(H(s)) = 2$; for each supernode t in any of these $H(s)$'s, $\lambda(H(t)) = 3$, and so on.

The DHs in our model are non-overlapping, which means that for all supernodes within a DH (i.e., across all levels), their corresponding nested dataflow graphs do not share any actors or edges. Furthermore, we assume that these DHs are finite, which means that the levels (λ values) are all bounded.

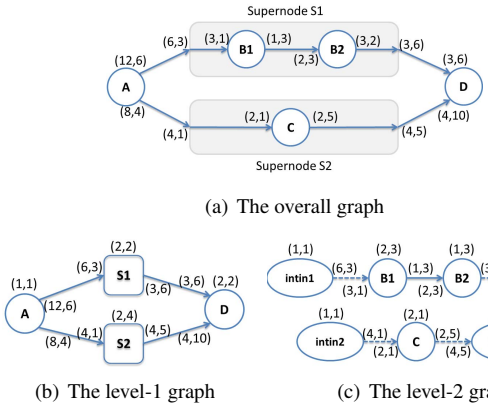
We refer to the maximum λ value in a DH D as the depth δ of D. For each $i \in \{1, 2, \ldots, \delta\}$, we denote by L_i the set of all actors that are "at level i". That is, $L_1 = V$, and for $i = 2, 3, \ldots, \delta$,

$$L_i = \cup\{V_h(s) | \lambda(H(s)) = i\}, \tag{2}$$

where $V_h(s)$ denotes the set of actors in the nested dataflow graph $H(s)$.

DHs in our decomposition approach can be constructed by designers as they explore alternative methods to structure the hierarchies such that they map efficiently into the parallelism hierarchy supported by the targeted platform. The key constraint in construction of a DH D is that the depth of each candidate DH should equal the platform depth. In Section 4.3, we illustrate how a DH can be constructed naturally from understanding of the flowgraph structure of an application. However, DHs can also be targeted by automated tools. Exploration of such automated DH construction tools is a useful topic for future work.

We have developed a systematic method, called *multidimensional DH mapping*, to specify and map these DHs into hierarchies of smaller graphs, which can be mapped to successively lower levels of the targeted platform hierarchy. Fig. 2 illustrates this approach for an MDSDF graph. The designer can construct the DHs bottom-up or top-down. At each ith level ($i > 1$) of the DH, one or more groups (*clusters*) of connected actors are combined into units that are viewed as individual supernodes from level $(i-1)$. Groups of actors, including supernodes, that are contained within such clusters are then scheduled together by adapting techniques for SDF- and MDSDF-based clustered graph analysis and scheduling [12, 3]. Use of these techniques to systematically derive production and consumption tuples associated with actors at different levels of the design hierarchy, as well as *firing vectors*, which determine the relative rates at which different actors in a cluster execute, is illustrated in Fig. 2.



(a) The overall graph

(b) The level-1 graph

(c) The level-2 graph

Fig. 2. An example of a DH for an MDSDF specification.

The actors labeled with the prefixes `intin` and `intout` in Fig. 2(c) represent *interface input* and *interface output actors* that are inserted based on the selected DH. These actors represent interfaces to the enclosing supernodes and serve to inject data from input edges and to output edges of the supernodes, while providing "standalone" dataflow graph representations for each level of the DH. Using these standalone representations, buffer management and scheduling are performed to ensure correct, consistent execution while mapping the actors in each DH level L_i into the corresponding ith level of the targeted DSP platform.

The production and consumption rates associated with the interface input and interface output actors are derived systematically using the cluster analysis techniques described above. Presently, we compute these rates by hand, as our emphasis in this work is on demonstrating the overall design methodology and its utility on a practical case study. However, the process can readily be automated since it is based on formal dataflow principles. Development of automated tool support for the design methodology developed in this paper is a useful direction for further work.

We omit further details of our multidimensional DH mapping approach in this paper due to space limitations. We demonstrate the utility of the approach in the next section with a case study of an important multidimensional signal processing subsystem, image histogram computation.

4. CASE STUDY

To demonstrate our proposed method for mapping MDSDF design hierarchies, we map an image processing application based on integral histogram computation [13] onto a GPU target platform.

The *integral histogram* (*IH*) first maps pixels into a set of non-overlapping ranges ("bins"), and then performs a 2-D scan. Two scan orders, cross-weave and wavefront, are

explored in [14]. The cross-weave scan processes the image in the first dimension (horizontal scan) followed by a scan in the second dimension (vertical scan). Instead of applying two passes, the wavefront scan propagates an anti-diagonal wavefront calculation as it operates through a single scan.

In our experiments, we incorporate use of a *tiled* image processing approach, where the image is separated into blocks (tiles) of neighboring pixels. Tiled approaches can be useful for GPU implementation to enhance parallel execution across multiple threads [4]. In particular, we explore in this case study a *tiled integral histogram* (*TIH*) approach for efficient mapping into GPU implementations.

The overall input image size for IH computation is denoted as $(I_w \times I_h)$ pixels, and the number of histogram bins is denoted as N_b. In TIH computation, an image is tiled as an $(N_w \times N_h)$ rectangular arrangement of tiles, where each tile has a $(T_w \times T_h)$ rectangular arrangement of pixels. Here, $T_w = I_w/N_w$, and $T_h = I_h/N_h$. For each $(T_w \times T_h)$ tile, the IH is calculated independently. After computation of all $(N_w \times N_h)$ tile-level IHs, the results can be processed to derive the image-level IH result.

We experiment with both tiled and non-tiled versions for the cross-weave scan. We have observed that non-tiled configurations of our wavefront-based IH actor perform with unacceptable latency on the targeted GPU, and therefore, we employ only tiled configurations when using the wavefront scan.

4.1. Actor Design

For GPU-based implementation of IH computation, we design three types of two-dimensional signal processing actors. These actors are parameterized so that they can be statically or dynamically configured (e.g., using parameterized dataflow [15] integration with MDSDF) for the desired type of IH computation. This parameterization in conjunction with our multidimensional DH mapping approach helps designers to explore trade-offs involving different IH computation strategies in conjunction with efficient parallel realizations of these strategies.

Each of the three actors employed in our IH case study has a single input port and a single output port. These actors are described as follows.

First, the **Bin-Check** actor determines bin membership for pixels. The actor executes pixel checks of an image column for all bins with $CONS = (1, I_h)$ and $PROD = (1, I_h \times N_b)$. Here, and in the remainder of this section, we denote the two-dimensional (MDSDF) production and consumption rates of a given actor port as $PROD$ and $CONS$, respectively.

Second, the **Intra-Tile-IH** actor computes the IH, where the size of the input tile is specified by the actor parameters T_w (width) and T_h (height), and the scan order is specified by the *scan order* parameter of the actor. The supported settings for the scan order parameter are:

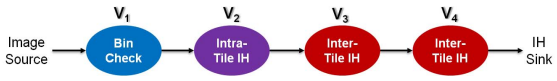


Fig. 3. MDSDF graph for optionally-tiled IH computation.

Table 1. Application modes.

App mode	Method	V2 SOP	V3 SOP	V4 SOP
APP-CWS	cross-weave TIH	CWS	HS	VS
APP-WFS	wavefront TIH	WFS	WFS	IDLE
APP-NT	no tiling	NT	IDLE	IDLE

- *CWS*: Compute the IH using a cross-weave scan with tiling. The actor ports satisfy $CONS = PROD = (T_w, T_h)$
- *WFS*: Compute the IH using a wavefront scan with tiling. The ports again satisfy $CONS = PROD = (T_w, T_h)$
- *NT*: Compute the IH using a cross-weave scan without tiling — that is, calculate the IH for the input image directly with $CONS = PROD = (T_w, T_h)$.

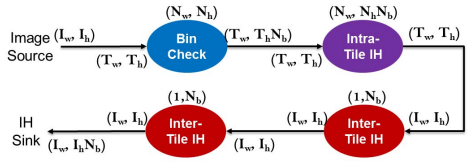
The **Inter-Tile-IH** actor performs accumulation among tiles with a parameter, called the *accumulation order* parameter, to support different scan orders for performing the accumulation. In particular, horizontal, vertical, and wavefront scans are used for accumulation order settings that are denoted HS, VS, and WFS, respectively. The actor ports of this actor (regardless of the accumulation order setting) satisfy $CONS = PROD = (I_w, I_h)$. In addition, the accumulation order parameter can be set to the value IDLE to bypass any accumulation. While in the IDLE configuration, the actor performs no computation, and simply passes its input to its output (through a simple pointer transfer to avoid memory transfer overhead).

4.2. Application Graph

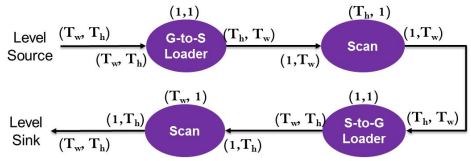
Given the actors developed in Section 4.1, one can implement the IH application with the MDSDF graph shown in Fig. 3. The desired scan orders and tiling settings can be achieved by setting the actor parameter values appropriately. In the experiments, we show performance comparisons among three specific application modes, which are defined by the groups of parameter settings shown in Table 1. Here, *SOP* stands for "scan order parameter."

4.3. DH Exploration

We customize the implementations for the different application modes by examining their MDSDF application graph representations separately, and deriving separate DHs to



(a) Grid level



(b) Block level for the Intra-Tile-IH actor.

Fig. 4. Hierarchical dataflow graphs for cross-weave TIH.

guide the application mapping processing. Taking the application mode labeled APP-CWS as an example, we show a DH in Fig. 4 that can be used to derive an efficient implementation on the targeted GPU. In the grid level of target platform parallelism, which is illustrated in Fig. 4(a), the 2-D indices shown above the actors represent the corresponding firing vectors that are derived from the DH (see Section 3). Each actor in the top level of the DH is mapped to a kernel function in the GPU, and the firing vector is used to configure the grid size.

Fig. 4(b) depicts the second level (i.e., block level) for the Intra-Tile-IH actor. Fig. 4(b) shows a hierarchical dataflow subgraph that specifies the internal functionality for the Intra-Tile-IH actor. To avoid non-coalesced memory access, the input data is loaded and transposed in the shared memory by the G-to-S Loader actor before the horizontal scan (G-to-S stands for "global-to-shared"). After the scan for each data row, the results are transferred from the shared memory back to the global memory by the S-to-G ("shared to global") Loader actor. Finally, a vertical scan is performed to obtain the IH for the input tile.

4.4. Experiments

In our experiments, an NVIDIA GTX260 GPU and an Intel Xeon 3GHz CPU are used. We compare the three different application modes in Table 1. Table 2 depicts the grid and block sizes for GPU kernels. Performance is compared for four image sizes ($I_w \times I_h$): 32x32, 64x64, 256x256, and 512x512. Based on the number of GPU threads employed for each kernel, we choose a tile size of (32×16) in the APP-CWS mode for all image sizes. For the APP-WFS mode, tile sizes of (4×4), (8×8), (16×8), and (32×16) are chosen for successively larger image sizes. We evaluate the frame processing time, including the time required for memory transfer from the host to the device (GPU) and the processing time on the device. We do not include the time for memory transfer from the device back to the host because many applications

Table 2. Grid sizes (upper) and block sizes (lower) derived from DH in our experiments.

mode	V2 kernel	V3 kernel	V4 kernel
APP-CWS	$(N_w, N_h N_b)$ $(T_w, 1)$	$(1, N_b)$ (T_w, T_h)	$(1, N_b)$ (T_w, T_h)
APP-WFS	$(1, N_b)$ (N_w, N_h)	$(1, N_b)$ (T_w, T_h)	N/A
APP-NT	$(1, N_b)$ $(I_w, 1)$	N/A	N/A

that employ IH can be implemented on the GPU efficiently without need for data transfer back to the CPU.

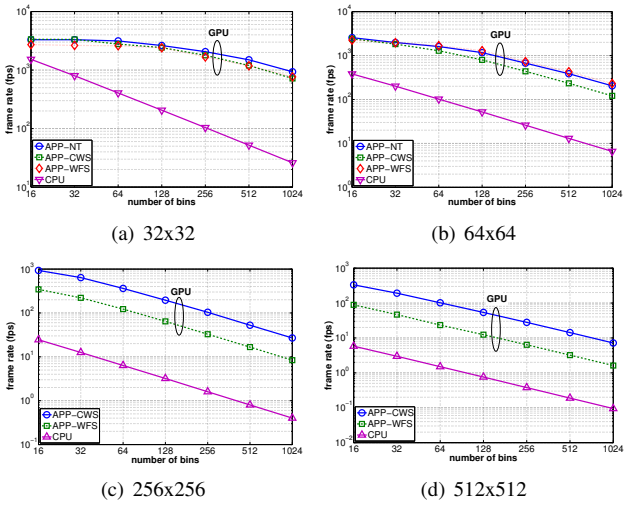
Fig. 5 shows the frame rates (i.e., $1/\tau$, where τ represents the average time in seconds required to process a single frame) for various bin sizes ranging from 16 to 1024. From the experimental results, we see that the GPU implementation of the IH consistently outperforms the CPU implementation, and that the speedup gains are approximately 35X for image sizes 32x32 and 64x64, 67X for image size 256x256, and 75X for image size 512x512.

Among the different GPU implementations for the 32x32 image size case, IH without tiling (APP-NT) provides the best performance since it avoids overhead from tiling. In the 64x64 case, however, APP-NT suffers from reduced inter-thread parallelism due to the large amount of shared memory required. The best performance is achieved in the APP-WFS mode, as this mode provides more threads in the V2 kernel and less overhead due to tiling (V4 is bypassed). With image sizes of 256x256 and 512x512, we must use tiling due to the size limitations of the shared memory. Compared to APP-WFS, APP-CWS can offer better frame rates by providing more effective parallel execution on the target platform.

In summary, the best application mode for IH calculation depends on the image size, and thus parameterized MDSDF application modeling in conjunction with our multidimensional DH mapping approach are useful design methods to map IH computations systematically onto the targeted GPU platform. Such a systematic mapping approach leads to designs that can be mapped more efficiently, and that are more portable, and easier to maintain and extend.

5. CONCLUSION

In this paper, we have developed a novel design method, building on the MDSDF model of computation, for hierarchical exploitation of parallelism in multidimensional signal processing applications. This method allows designers to explore alternative implementations in a manner that separates platform-specific parallel processing optimization from the behavioral specification, thereby enhancing portability and trade-off exploration. More specifically, our multidi-



(a) 32x32 (b) 64x64

(c) 256x256 (d) 512x512

Fig. 5. Performance comparisons for different image sizes.

mensional design hierarchy model provides an intermediate model that provides a formal linkage between hierarchical layers of parallelism in the target platform and corresponding subsystems of the application that will be mapped onto these layers. In our approach, graph clustering and MDSDF dataflow analysis are applied in novel ways to map applications to target platforms that employ parallelism at multiple levels. Experimental results show that fast GPU implementations can be derived from the approach, as well as efficient trade-off analysis and optimization across different application modes.

6. ACKNOWLEDGEMENT

The authors acknowledge the Government's support in the publication of this paper. The material is based upon work funded by the US Air Force Research Laboratory (AFRL), and Laboratory for Telecommunication Sciences (LTS). Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not reflect the views of AFRL or LTS.

7. REFERENCES

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

[2] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.

[3] P. K. Murthy and E. A. Lee, "Multidimensional synchronous dataflow," *IEEE Transactions on Signal Processing*, vol. 50, no. 8, pp. 2064–2079, August 2002.

[4] *NVIDIA CUDA C Programming Guide*, April 2012, Version 4.2.

[5] G. Roquier, M. Wipliez, M. Raulet, J. W. Janneck, I. D. Miller, and D. B. Parlour, "Automatic software synthesis of dataflow program: An MPEG-4 simple profile decoder case study," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, October 2008.

[6] S. Kwon, H. Jung, and S. Ha, "H.264 decoder algorithm specification and simulation in simulink and PeaCE," in *Proceedings of the International SoC Design Conference*, October 2004, pp. 9–12.

[7] J. Keinert, C. Haubelt, and J. Teich, "Modeling and analysis of windowed synchronous algorithms," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, May 2006.

[8] P. Boulet, "Array-OL revisited, multidimensional intensive signal processing specification," Tech. Rep., INRIA, February 2007.

[9] D. Ko and S. S. Bhattacharyya, "Modeling of block-based DSP systems," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 40, no. 3, pp. 289–299, July 2005.

[10] 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.

[11] *ATI Stream Computing OpenCL Programming Guide*, June 2010.

[12] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*, Kluwer Academic Publishers, 1996.

[13] F. Porikli, "Integral histogram: a fast way to extract histograms in cartesian spaces," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005, pp. 829–836.

[14] P. Bellens, K. Palaniappan, R. M. Badia1, G. Seetharaman, and J. Labarta, "Parallel implementation of the integral histogram," in *Proceedings of the International Conference on Advanced Concepts for Intelligent Vision Systems*, 2011.

[15] 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.