# MODELING AND OPTIMIZATION OF BUFFERING TRADE-OFFS FOR HARDWARE IMPLEMENTATION OF IMAGE PROCESSING APPLICATIONS

*Dong-Ik Ko and Shuvra S. Bhattacharyya*

{dik, ssb}@eng.umd.edu

Department of Electrical and Computer Engineering, and Institute for Advanced Computer Studies,
University of Maryland, College Park, 20742, USA.

## ABSTRACT

As modern image and video processing applications handle increasingly higher image resolutions, the buffering requirements between communicating functional modules increase correspondingly. The performance and cost of these applications can change dramatically depending on the implementation methods for FIFO buffers and the data delivery methods between modules. This paper introduces a new FIFO hardware mapping algorithm based on pointer-based token delivery from dataflow semantics for image and video processing applications. This approach significantly improves the performance of dataflow based implementation of image and video processing systems, and allows effective prediction of changes in performance and buffer memory requirements associated with changes in image resolution. Our pointer-based token delivery method allows indirect token delivery between actors by pointers in conjunction with use of a shared memory. Each pointer references a data block stored in the shared memory. In pointer-based token delivery, a buffer can be configured to be implemented as the combination of a small, fast FIFO and a larger, relatively cheap shared memory while providing an attractive trade-off between performance and hardware cost. We present the complete semantics of our pointer-based modeling method, systematic techniques for mapping representations using these semantics into efficient implementations, and experimental results that demonstrate the performance of the proposed pointer-based techniques.

## 1. RELATED WORK

Dataflow [7] is widely used for designing DSP applications. Various research efforts on mapping dataflow graphs into hardware implementations have been undertaken. For example, the approach of [2] exploits loop parallelism to map nested loop kernels onto a coarse-grained reconfigurable architecture. The approach of [3,4] uses direct mapping of each dataflow graph component (actor) onto a corresponding hardware resource. The approach of [5] uses shared resources and looped schedules. The approach of [6] analyzes a given set of applications to extract commonalities across nodes in different applications and uses them to bias the mapping of nodes in the partitioning process. For FPGA implementation, the approach of [10] provides a rapid system prototyping method through a component architecture and an associated set of software tools. The approach of [11] provides a pipelined asynchronous circuit mapping method. For pointer synthesis, the approach of [9] encodes pointer values and generates circuits that can dynamically access different locations with each pointer reference. The approach of [13] points out that pointers can reference indices to RAM, registers or even wires in a hardware mapping. The approach of [1] applies an external memory for mapping FIFO buffers and implements real-time image convolution on an FPGA. The approach of [8] implements image processing applications on FPGAs and points out that such implementations lead to a large on-chip FIFO buffers that prevent flexible usage of FPGAs for image processing applications. The approach of [12] presents an elaborate technique for mapping global, static arrays to distributed communication structures while classifying four types of inter-process communication patterns. The approach of [15] studies memory optimization for embedded software, particularly the performance of cache-based systems. The approach of [14] presents a novel technique for background memory allocation in multi-dimensional signal processing applications based on dataflow analysis.

The efforts described above make useful contributions to mapping application representations at various levels of abstraction into hardware implementations. However, the simultaneous analysis of both performance and cost implications when mapping image processing applications, which involve especially large volumes of data token delivery, has not been thoroughly investigated in previous work.

This paper helps to bridge this gap by studying, in the context of mapping dataflow graphs into hardware, the relationship between token delivery methods (indirect, pointer-based token delivery vs. direct-reference, raw token delivery) and FIFO architecture. This paper exploits pointer-based token delivery to reduce on-chip FIFO sizes, and also provides a range of efficient trade-offs between performance (latency and throughput) and FPGA resource cost through a novel FIFO mapping algorithm. This paper also shows how overall performance and cost vary in relation to the selected sub-frame size at which block processing is carried out. Finally, this paper provides a new mapping algorithm for dataflow representations of image processing applications to reduce overall FPGA resource costs without significant performance loss.

## 2. FIFO HARDWARE MAPPING FOR DATAFLOW GRAPHS

### 2.1 Modeling and architecture

In this work, an application is modeled under synchronous dataflow (SDF) [7] semantics and then mapped to an FPGA device. Each vertex (actor) within the given SDF graph is mapped to a module within the target FPGA. Edges are con-



Authorized licensed use limited to: University of Maryland College Park. Downloaded on March 14, 2009 at 16:36 from IEEE Xplore. Restrictions apply.

verted into either pure on-chip *raw data FIFO* architectures or composite FIFO architectures that we call *pointer based FIFOs*. Figure 1 shows a comparison of raw data FIFOs and pointer based FIFOs. In Figure 1b), the raw data FIFO is embedded inside the FPGA chip and holds direct raw data tokens. Here, by *token* we mean the unit of data transfer along an edge in the dataflow graph. The pointer based FIFO involves both an on-chip FIFO, which holds references to token blocks rather than the tokens themselves, and an external (off-chip) RAM-based memory, which may be shared across multiple pointer based FIFOs as well as other storage constructs. In Figure 1a), raw data tokens are located in the external memory, while a relatively small on-chip FIFO buffer holds pointers that provide a stream of indices into the external memory.

The FIFO architectures (raw data vs. pointer based) and FIFO sizes can be configured strategically based on optimization during the synthesis process. This paper formulates and investigates this optimization problem, and studies various important factors that should be taken into account when configuring dataflow buffers for hardware mapping. This is an important problem because the configurations of the FIFOs in a dataflow graph implementation have significant impact on the overall performance and hardware resource costs. This paper presents an effective heuristic FIFO mapping algorithm for mapping SDF graphs efficiently into hardware.

### 2.2 Performance and cost impact of token delivery methods

As implied above, we consider two alternative token delivery methods between dataflow actors, pointer based token delivery (indirect token delivery) and raw token delivery (direct token delivery).

Raw token delivery is the conventional form of data delivery for dataflow graph implementation. Raw token delivery directly transfers data tokens across the FIFOs that connect adjacent pairs of actors in the dataflow graph. Therefore, for applications, such as those found in the image processing domain, that require large volumes of token transfer, very high resource requirements often result from extensive use of raw token delivery. On the other hand, since there is no indirection overhead or external memory access involved, raw token delivery improves performance through faster dataflow communication.

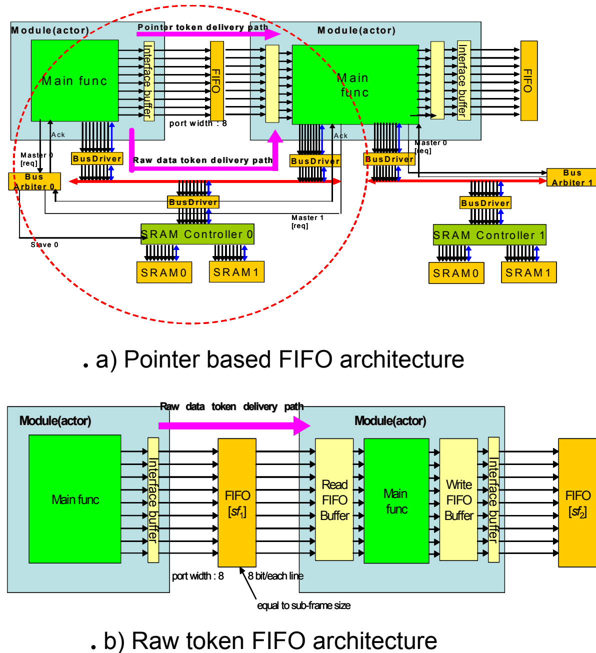
The limited quantities of gates available on FPGAs makes it challenging to implement image processing applications efficiently on these devices. Although FPGA resource density continues to increase from Moore's law, the complexity and resolution requirements of state-of-the-art image processing applications is also increasing at a significant pace.

Pointer based token delivery allows for more efficient use of limited FPGA resources by dividing inter-actor communication functionality into two parts. These parts consist of a relatively small set of pointers, and blocks of token data that the pointers reference. The pointers are kept in fast but expensive on-chip FIFOs, while the raw token data is located in slow but cost-effective external RAM. Dataflow graph actors send data to other actors by transferring pointers through the on-chip FIFOs. Actors at the receiving end use the transferred pointers to access external memory and retrieve the actual raw tokens. Pointer based token delivery significantly reduces FPGA resource requirements at the expense of some degradation in latency and throughput.

Equation (1) below describes relationships between pointer based token delivery and raw token delivery in terms of performance (execution time) and cost (the required number of gates). Here, $g_F$ denotes the number of gates required for the FIFO $F$; $t_F$ denotes the execution time for data token delivery through FIFO $F$; $\alpha_g$ represents a coefficient for converting the number of gates between two delivery methods; and $\alpha_t$ represents a similar conversion coefficient for execution time. The values of $\alpha_g$ and $\alpha_t$ depend on the subframe size $sf$.

$$g_{raw} \gg g_{ptr} \ , \ g_{raw} = \alpha_g g_{ptr} \ , \ t_{Fraw} \ll t_{Fptr} \ ,$$
$$t_{Fptr} = \alpha_t t_{Fraw} \ . \qquad (1)$$

The following equation describes the effects of raw token delivery and pointer based token delivery on latency and throughput:




. a) Pointer based FIFO architecture

. b) Raw token FIFO architecture

**Figure 1.** Comparison of FIFO architectures

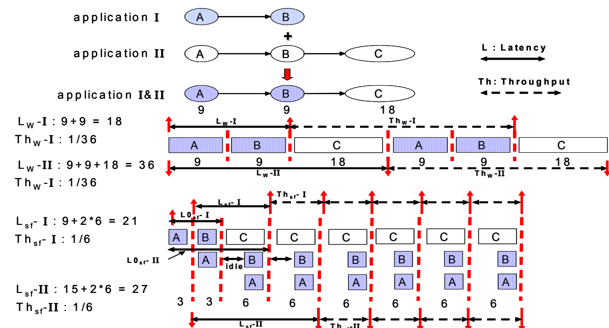



**Figure 2.** Effect of sub-frame division on latency and throughput.



Authorized licensed use limited to: University of Maryland College Park. Downloaded on March 14, 2009 at 16:36 from IEEE Xplore. Restrictions apply.

$$L = \sum_{i=1}^{n} [t_A(a_i) + t_O(a_i) \qquad . \qquad (2)$$
$$+ \beta_i^{\text{in}} t_{\text{Fraw}}(a_i) + (1 - \beta_i^{\text{in}}) t_{\text{Fptr}}(a_i)$$
$$+ \beta_i^{\text{o}} t_{\text{Fraw}}(a_i) + (1 - \beta_i^{\text{o}}) t_{\text{Fptr}}(a_i)]$$

Here, a critical path of the given application must be extracted beforehand for the analysis, and $n$ is the number of actors on this critical path. The symbols $\beta_i^{\text{in}}$ and $\beta_i^{\text{o}}$ are related, respectively, to the input port and output port of $a_i$ in the critical path (i.e., with respect to the edges in the critical path that are incident to $a_i$). In (2), $\beta_i^{\text{in}} = 1$ ($\beta_i^{\text{o}} = 1$) if the associated communication is mapped to a raw FIFO architecture, and conversely, $\beta_i^{\text{in}} = 0$ ($\beta_i^{\text{o}} = 0$) if it is mapped to a pointer based FIFO. The other symbols in (2) are defined below in Section 2.3.

### 2.3 Effect of sub-frame size on performance and cost

Sub-frame division reduces FIFO size along with pointer based token delivery since the whole data frame can be processed in smaller units. However, depending on the application, there may be strict constraints on the sub-frame size ($sf$) that can be employed. Many image processing subsystems have minimum window (or block) sizes for their basic units of operation. Some globally-oriented operations, such as contouring, require the whole image frame as their basic units of input.

Sub-frame division influences both performance and cost. To understand this better, we can decompose the execution time of an actor $a_i$ into three different parts, $t_A(a_i)$, $t_O(a_i)$ and $t_F(a_i)$. Here, $t_A(a_i)$ is the execution time for activation of $a_i$; $t_O(a_i)$ is the execution time for the main functional logic operation of $a_i$; and $t_F(a_i)$ is the execution time required for token delivery of $a_i$. $t_A(a_i)$ is proportional to the number of sub-frame divisions ($\delta$), whereas the "total summation" of $t_O(a_i)$ and $t_F(a_i)$ are the same regardless of the sub-frame division format. Usually, $t_A(a_i)$ is relatively small compared to $t_O(a_i)$ and $t_F(a_i)$.

Equation 3 shows the relationship among the three different components of execution for an actor, taking into account sub-frame division.

$$L_w(a_i) = t_{A_w}(a_i) + t_{O_w}(a_i) + t_{F_w}(a_i) ,$$
$$L_{sf}(a_i) = \delta t_{A_{sf}}(a_i) + \delta[t_{O_{sf}}(a_i) + t_{F_{sf}}(a_i)] ,$$
$$L_w(a_i) \le L(a_i) \le L_{sf}(a_i) ,$$
$$t_{O_w}(a_i) + t_{F_w}(a_i) \cong \delta[t_{O_{sf}}(a_i) + t_{F_{sf}}(a_i)] ,$$
$$t_{A_w}(a_i) = t_{A_{sf}}(a_i) ,$$
$$t_{A_w}(a_i) < \delta t_{A_{sf}}(a_i) \ll t_{O_w}(a_i), t_{F_w}(a_i) . \qquad (3)$$

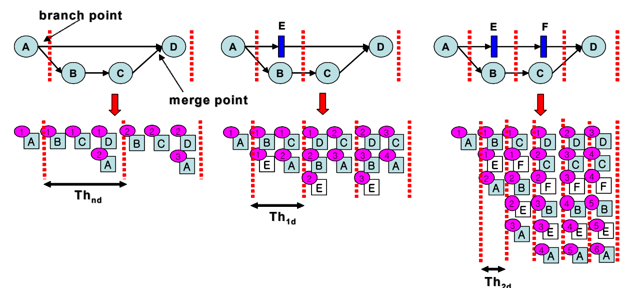
Here, $w$ represents the size of the entire image frame; $sf$ is the sub-frame size; $\delta$ is the number of sub-frame divisions ($\delta = w/\text{sf}$); and $L(a_i)$ is latency of actor $a_i$. Additionally, $L_w(a_i)$ and $L_{sf}(a_i)$ are latencies of actor $a_i$ under the image frame size $w$ and under the sub-frame size $sf$, respectively. Unlike the latency and throughput of a single actor, as decomposed in (3), the latency and throughput of the entire application are influenced by the interaction of data dependency, sub-frame size and FIFO architecture. Although sub-frame division generally allows for reduction of FIFO size, and also improves throughput, sub-frame division generally leads to some increase in application latency. For example, in the case where a single dataflow graph represents two or more applications operating concurrently, and those applications share actors in the graph, data dependencies and execution time distributions of paths in the graph influence the performance of each application in the dataflow graph differently.

Figure 2 compares, for an illustrative example, the performance of sub-frame division by $\delta = 3$ to the case where there is no sub-frame division. Here, throughput is improved for both Applications I and II. However, sub-frame division degrades the latency of Application I, whereas the latency of Application II is improved. This phenomenon generally arises when two or more applications share actors (e.g., for more compact representation and implementation) in a common dataflow graph and $\rho$ (defined in (4) below) is smaller than 0. This effect becomes prominent especially when the ratio of $idle$ and $L_W$ is large, where $idle$ represents the pipeline idle time. In (4), $L_o$ can be obtained by simply dividing $L_W$ by $\delta$.

$$\rho = \delta L_o - \left[L_o + idle + \frac{(\delta - 1)}{Th}\right] \qquad . \qquad (4)$$
$$= (\delta - 1)\left(L_o - \frac{1}{Th}\right) - idle$$

$$L_w = \delta L_o ,\ Th_w = 1/L_w ,\ L_{sf} = L_1 + \frac{(\delta - 1)}{Th_{sf}} ,$$
$$Th_{sf} = \frac{1}{actor_{\max}} ,\ \text{Always, } Th_{sf} < Th_w ,$$
$$\rho > 0 \rightarrow L_w \ge L_{sf}, \text{ otherwise} \rightarrow L_w < L_{sf} \qquad . \qquad (5)$$

In (5), $actor_{\max}$ is the execution time of the actor with the largest execution time, and $L_1$ represents the initial latency for subframe size $sf$. Here, the number of gates required for each application ($g(Appl)$) in the common graph is reduced by increasing $\delta$. Equation (6) shows the effect of sub-frame division on the number of gates required for an application($Appl$):




. a) 0 delay FIFO . b) 1 delay FIFO . c) 2 delay FIFO

**Figure 3.** Effect of data dependency on performance.



Authorized licensed use limited to: University of Maryland College Park. Downloaded on March 14, 2009 at 16:36 from IEEE Xplore. Restrictions apply.

$$g_{\mathrm{w}}(Appl) \approx \delta g_{\mathrm{sf}}(Appl) \ . \qquad (6)$$

### 2.4 Effect of data dependency on performance and cost

In case a dataflow graph has a "branch point", two or more paths following the branch point merge again at some subsequent point, and these paths exhibit a large execution time deviation, the associated data dependency can greatly deteriorate the performance of all the associated applications in the dataflow graph. Here, a "branch point" represents a point where a single actor has two or more output ports or a single output port goes to two or more successor actors.

Figure 3 shows how performance under sub-frame division can be improved through insertion of special FIFOs that we call "delay FIFOs( $F_{\mathrm{delay}}$ )" (these are the FIFOs labeled $E$ and $F$ in Figure 3). Performance improvement by delay FIFO insertion depends on the execution time distribution of the actors on each critical path following the branch point.

Equation (7) represents the relationship between performance and the added delay FIFOs.

$$L_{\mathrm{nd}} = L_{\mathrm{nd,1}} + \frac{(\delta - 1)}{Th_{\mathrm{nd}}} \ , \ L_{\mathrm{1d}} = L_{\mathrm{1d,1}} + \frac{(\delta - 1)}{Th_{\mathrm{1d}}} \ ,$$

$$L_{\mathrm{2d}} = L_{\mathrm{2d,1}} + \frac{(\delta - 1)}{Th_{\mathrm{2d}}} \ , \ L_{\mathrm{nd,1}} = L_{\mathrm{1d,1}} = L_{\mathrm{2d,1}} \ ,$$

$$Th_{nd} < Th_{1d} < Th_{2d} \ , \ \therefore L_{nd} > L_{1d} > L_{2d} \qquad (7)$$

Here, $L_{\mathrm{nd}}$ and $Th_{\mathrm{nd}}$ are the latency and throughput, respectively, without $F_{\mathrm{delay}}$ . Furthermore, $L_{\mathrm{1d}}$ and $Th_{\mathrm{1d}}$ are the corresponding values with 1 $F_{\mathrm{delay}}$ and $L_{\mathrm{2d}}$ and $Th_{\mathrm{2d}}$ are those for 2 $F_{\mathrm{delay}}$ s. $L_{\mathrm{nd,1}}$ , $L_{\mathrm{1d,1}}$ and $L_{\mathrm{2d,1}}$ are latencies for processing the first subframe in the cases of no $F_{\mathrm{delay}}$ , 1 $F_{\mathrm{delay}}$ and 2 $F_{\mathrm{delay}}$ s, respectively.

Equation (8) represents the increase in the number of gates required for the application as delay FIFOs are added. The overhead of the delay FIFOs can be minimized by using the pointer based FIFO architecture for their implementation.

$$g_{\mathrm{nd}}(Appl) \le g_{\mathrm{1d}}(Appl) \le g_{\mathrm{2d}}(Appl) \ . \qquad (8)$$

### 2.5 Optimization of FIFO hardware mapping

Idle intervals and uneven execution time distributions exist due to data dependencies and differences in operational complexity across dataflow actors. Performance and cost can be improved by integrating cost-effective, pointer based FIFOs and fast, raw token FIFOs in strategic ways.

Figure 4 provides a simple illustration of how the resource cost $g(G)$ for a dataflow graph $G$ can be reduced significantly while maintaining overall performance through hybrid FIFO architecture selection. Here, the throughputs of both configurations are identical. Furthermore, by using sub-frame division, the difference between latency of Figures 4a and 4b can be made negligible, since the throughput ( $Th$ ) is ultimately the primary factor for determining latency under sub-frame division, as implied by (4) and (5).

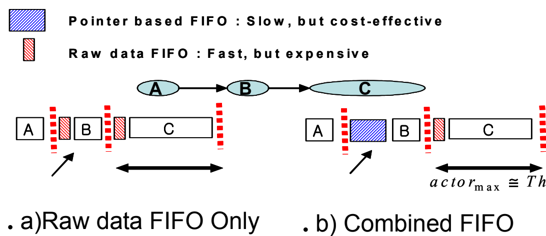



**Figure 4.** Comparison of FIFO mapping.

Figures 5 and 6 show our FIFO mapping algorithm, which is motivated by the observations and analysis above. It is assumed that the dataflow graph $G$ can involve multiple applications, and moreover, that subsets of applications can share common actors for more compact representation and implementation. The function $initializeGraph()$ sets up information about estimated execution times and execution time distributions of the actors. The function also finds $g_{logic}(G)$ and $g_{\mathrm{Fraw}}(G)$ . Here, $g_{logic}(G)$ represents the estimated number of gates for the main functional logic portions the actors, and $g_{\mathrm{Fraw}}(G)$ is the number of gates used for FIFOs under the assumption that only raw token FIFOs are used. The actual $g(G)$ that results from a mapped implementation lies between $g_{\mathrm{min}}$ and $g_{\mathrm{max}}$ as shown in (9).

$$g_{\mathrm{min}}(G) = g_{\mathrm{logic}}(G) + g_{\mathrm{Fptr}}(G) \ ,$$

$$g_{\mathrm{max}}(G) = g_{\mathrm{logic}}(G) + g_{\mathrm{Fptr}}(G) + g_{\mathrm{Fdelay}}(G) \ ,$$

$$g_{\mathrm{min}}(G) \le g(G) \le g_{\mathrm{max}}(G) \ . \qquad (9)$$

For each application( $G_{cur}[i]$ ), a critical path ( $G_{\mathrm{curHg}}[i]$.crPath ) is selected and an appropriate FIFO type is determined based on the execution time distribution of actors within the path.

For each hierarchical subsystem within the critical path, $detFIFOArch()$ is applied recursively. Finally, delay FIFO ( $F_{\mathrm{delay}}$ ) insertion is performed to improve performance. For $F_{\mathrm{delay}}$ , pointer based FIFOs ( $F_{\mathrm{ptr}}$ ) are used, and therefore, the overhead of redundant FIFOs can be minimized while achieving the desired performance improvement.

## 3. EXPERIMENTAL RESULTS

Figure 7 shows a complex, composite morphological image processing application used in this paper for experimentation. Here, the performance and cost of each application under the dataflow representation are influenced by the interaction of to shared actors with the applications that contain them. Figure 7 is implemented by Verilog and is simulated under the modelSim 6.0$a$ environment. Synthesis is performed under Xilinx XST with the Spartan3 (xc3s1500) used as the target device. Input images of size ( $w = 128 \times 128$ ) are consumed and processed by the graph. Experimentation is performed under two different values of $sf$, corresponding to 8x8 and 16x16 subframes. In Table 1, $C1$ and $C4$ of $F_{\mathrm{raw}}$ are lower bounds in performance optimization, and $C2$ and $C5$ of $F_{\mathrm{ptr}}$ are lower bounds in cost reduction. Equation (10) shows how, in the following discussion, we compare the performance and costs of two different configurations $C_X$ and $C_Y$.

$$[\sum L_{C_X} / \sum L_{C_Y} + \mathrm{Th}_{C_X} / \mathrm{Th}_{C_Y}]/2 \ , \ \mathrm{g(G}_{C_X}) / \mathrm{g(G}_{C_Y}) \ . \qquad (10)$$

In comparison of $F_{\mathrm{raw}}$ and $F_{\mathrm{ptr}}$ , $C1$ and $C4$ provide



Authorized licensed use limited to: University of Maryland College Park. Downloaded on March 14, 2009 at 16:36 from IEEE Xplore. Restrictions apply.

approximately 23% performance improvement compared with $C2$ and $C5$, while requiring about 81% more gates. In comparison of $F_{\text{delay}}$, $C6$ provides 54% performance improvement compared with $C3$ along with a slight (2%) cost increase. In comparison of sub-frame division effects for $C4$, $C5$ and $C6$, the latency of *Smoothing* is slightly improved, whereas the latency of *Gradient* is decreased as $sf$ is decreased. Here, the latency impact is negligible since *idle* is relatively small compared to the execution time of each actor for processing the entire image frame $w$. On the other hand, the throughput and cost improvements are distinguishable as $\delta$ is increased.

Next, we see that $C6$, which involves both performance and cost optimization, provides 54% performance improvement and 16% cost reduction compared with the conventional approach of $C1$. Similarly, $C5$, which leans more toward cost optimization, provides 39% performance improvement and 76% cost reduction compared with the conventional approach of $C1$. Here, delay FIFO insertion in Path 1 (see Figure 7) leads to significant improvement of performance along with 2% increase of $g(G)$. Combined use of $F_{\text{ptr}}$ and $F_{\text{raw}}$ with $F_{\text{delay}}$ significantly improves overall performance along with providing for cost reduction. For cases where cost is the primary issue, it is important to note the significant cost reduction of $F_{\text{ptr}}$.

## 4. CONCLUSIONS AND FUTURE WORK

This paper studies important issues in the mapping of dataflow representations of image processing applications into hardware implementations. Specifically, we focus on efficient mapping of FIFO buffers, and explore the effects of FIFO architecture, sub-frame division and data dependency on performance and cost. Based on this exploration, we provides heuristic optimization methods that simultaneously improve performance and cost with manageable complexity. A strategic FIFO mapping approach that comprehensively exploits dataflow graph characteristics results in significantly lower FPGA resource requirements with nearly equal performance. Useful directions for future work include

```
initializeGraph(G) {
    — Analyze the critical path of each application in
  the dataflow graph.
    — Obtain the estimated execution time
    — Obtain the execution time distribution on the path
    — Obtain g_logic(G) and g_Fraw(G)
    return g_logic(G) , g_Fraw(G) ;
  }
detSubFrameDivision(G){
    if(L_w(G.appl_highest_priority)
     < L_sf(G.appl_highest_priority))        {
       dataFrame = w ;
    }
    else {
       dataFrame = sf ;
    }
   — apply all other applications with  dataFrame
}
```

**Figure 5.** FIFO mapping algorithm-PartA.

```
detFIFOArch(G){
    g_Fraw(G), g_logic(G) =initializeGraph(G);
    g_F_G = 0 ;
    while(G ≠ ϕ){
      - Select an application( G_curHg ) of highest
      priority
      - while( G_curHg[i] ≠ ϕ ){
          g_F_path =detFIFOType( G_curHg[i].crPath );
          g_F_hier = 0 ;
          for each hier actor Φ[j] of  G_curHg[i].crPath  {
             g_F_hier = g_F_hier +detFIFOArch(Φ[j] ;
          }
            G_curHg = G_curHg – G_curHg[i].crPath  ;
           g_F_G = g_F_G + g_F_path + g_F_hier ;
      }
       G = G – G_curHg ;
    }
    if(g_F_raw(G) > g_F_G) {
      - Perform data dependency analysis of G
      - Insert delay FIFOs( F_delay : F_ptr type)
      and update g_F_G
        g_F_G = g_F_G + g_Fdelay ;
    }
    detSubframeDivision(G);
    return   g_F_G + g_logic(G)  ;
}

detFIFOType(G){
    g_F_sum =0;
    for each actor on the G {
      if([t_O(G.a[i]) + t_Fptr(G.a[i])]
        < [t_O(actor_max) + t_F(actor_max)]{)
          g_F_sum = g_F_sum + g_F_ptr(a[i]) ;
      }
      else {
          g_F_sum = g_F_sum + g_F_raw(a[i]);
      }
    }
    return  g_F_sum  ;
}
```

**Figure 6.** FIFO mapping algorithm-PartB.



Authorized licensed use limited to: University of Maryland College Park. Downloaded on March 14, 2009 at 16:36 from IEEE Xplore. Restrictions apply.

extending the methodology developed in this paper to heterogeneous, embedded multiprocessors that include a variety of processing components, such as conventional FPGAs, platform FPGAs, and programmable digital signal processors.

## 5. REFERENCES


[1] A. Benedetti, A. Prati, N. Scarabottolo, "Image Convolution on FPGAs: The Implementation of a Multi-FPGA FIFO Structure," 24 th. EUROMICRO Conference Volume 1 (EUROMICRO'98), August 25 - 27, 1998.

[2] F. Hannig, H. Dutta and J. Teich, "Regular Mapping for Coarse-grained Reconfigurable Architectures", In Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004), Vol. V, pp. 57-60, Montreal, Quebec, Canada, May 17-21, 2004.

[3] J. Horstmannshoff, T. Grotker, H. Meyr "Mapping multirate dataflow to complex RT level hardware models", IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '97), July 14 - 16, 1997 Zurich, SWITZERLAND.

[4] J. Horstmannshoff, H. Meyr, "Efficient building block based RTL code generation from synchronous data flow graphs", Annual ACM IEEE Design Automation Conference Proceedings of the 37th conference on Design automation, Los Angeles, California, United States, Pages: 552 - 555, 2000.

[5] H. Jung, S. Ha, "Hardware Synthesis from Coarse-Grained Dataflow Specification for Fast HW/SW Cosynthesis", International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'04), September 08 - 10, 2004.

[6] A. Kalavade and P. A. Subhrahmanyam, "Hardware / Software Partitioning for Multi-function Systems", Proc. International Conference on Computer Aided Design, pp. 516-521, Nov. 1997.

[7] E.A. Lee, D.G. Messerschmitt, "Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing", *IEEE Transactions on Computers*, February, 1987.

[8] A. E. Nelson, "Implementation of Image Processing Algorithms on FPGA hardware", MS Thesis, Vanderbilt University, May 2000.

[9] Luc Semeria, K. Sato, G. Micheli, "Synthesis of hardware models in C with pointers and complex data structures", IEEE Transactions on Very Large Scale Integration (VLSI) Systems archive Volume 9, Issue 6 (December 2001), Pages: 743 - 756

[10] G. Spivey, S. S. Bhattacharyya, K. Nakajima, "A Component Architecture for FPGA-based, DSP System Design", In Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors, San Jose, California, July 2002.

[11] J. Teifel and R. Manohar. An Asynchronous Dataflow FPGA Architecture. IEEE Transactions on Computers, Special issue on Field-Programmable Logic, November 2004.

[12] A. Turjan, B. Kienhuis, Ed F. Deprettere: An Integer Linear Programming Approach to Classify the Communication in Process Networks. SCOPES 2004: 62-76

[13] N. Vanspauwen, E. Barros, S. Cavalcante, C. Valderrama, "On the Importance, Problems and Solutions of Pointer Synthesis", Proceedings of the 15th symposium on Integrated circuits and systems design, pp: 317, 2002.

[14] I.Verbauwhede, F.Catthoor, J.Vandewalle, H.De Man, "Background memory management for the synthesis of algebraic algorithms on multi-processor DSP chips", Proc. VLSI'89, Int. Conf. on VLSI, Munich, Germany, pp.209-218, Aug. 1989.

[15] W. Wolf and M. Kandemir, "Memory system optimization of embedded software," Proceedings of the IEEE, 91(1), January 2003, pp. 165-182.

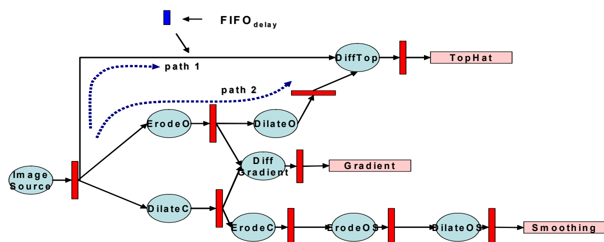




**Figure 7.** Complex, composite morphological image processing application (TopHat, Gradient and Smoothing).

Table 1. Comparison of FIFO mapping results.

| | | No delay | | |
|---|---|---|---|---|
| sf(8x8), $\delta$=256 | | C1 | C2 | C3 |
| L | $L_s$ | 888 us | 1157us | 888 us |
| | $L_g$ | 884 us | 1152us | 884 us |
| | $L_t$ | 885 us | 1154us | 886 us |
| Th | | 3.5 us | 4.5us | 3.5 us |
| g(G) | | 122,915 | 26,840 | 101,565 |
| sf(16 x16), $\delta$=64 | | C1 | C2 | C3 |
| L | $L_s$ | 886 us | 1158 us | 886 us |
| | $L_g$ | 869 us | 1136 us | 871 us |
| | $L_t$ | 876 us | 1144 us | 878 us |
| Th | | 14 us | 18 us | 13.5 us |
| g(G) | | 562,793 | 26,969 | 443,721 |

| | | Delay FiFo | | |
|---|---|---|---|---|
| sf(8x8), $\delta$=256 | | C4 | C5 | C6 |
| L | $L_s$ | 447 us | 581us | 447 us |
| | $L_g$ | 443 us | 576us | 443 us |
| | $L_t$ | 445 us | 578us | 445 us |
| Th | | 1.5 us | 2us | 1.5 us |
| g(G) | | 125,516 | 29,441 | 104,166 |
| sf(16 x16), $\delta$=64 | | C4 | C5 | C6 |
| L | $L_s$ | 455 us | 594 us | 455 us |
| | $L_g$ | 437 us | 572 us | 439 us |
| | $L_t$ | 444 us | 581 us | 446 us |
| Th | | 6 us | 8 us | 6 us |
| g(G) | | 565,403 | 29,579 | 446,331 |

- L: Latency, Th: Throughput.
- $g(G)$ : the number of gates for graph $G$ .
- C1, C4: $F_{raw}$ , C2, C5: $F_{ptr}$ , C3, C6: $F_{raw} + F_{ptr}$ .



Authorized licensed use limited to: University of Maryland College Park. Downloaded on March 14, 2009 at 16:36 from IEEE Xplore. Restrictions apply.