Automated Hybrid Interconnect Design for FPGA Accelerators Using Data Communication Profiling

Cuong Pham-Quoc, Zaid Al-Ars, Koen Bertels
Computer Engineering Lab, Delft University of Technology
Email: {P.PhamQuocCuong,Z.Al-Ars,K.L.M.Bertels}@tudelft.nl

***Abstract*—In this paper, we introduce an automated interconnect design strategy to create an efficient custom interconnect for kernels in an FPGA-based accelerator system to accelerate their communication behavior. Our custom interconnect includes an NoC, shared local memory solution or both. Depending on the quantitative communication profiling of the application, the interconnect is built using our proposed custom interconnect design algorithm and adaptive mapping function. Experimental results show that our system achieves an overall application speed-up of $3.72\times$ compared to software and of $2.87\times$ compared to the baseline system - a conventional FPGA bus-based accelerator system. Moreover, our proposed system achieves 66.5% energy reduction due to the reduced execution time.**

***Index Terms*—FPGA-based accelerator, communication profiling, custom interconnect.**

I. Introduction

Although FPGAs offer a high-performance and energy-efficient computing compared to a general purpose processor (GPP), it is not easy and not straightforward to develop a complex application using only FPGAs. FPGA-based accelerators represent an approach to use the advantages of both the FPGA and the GPP. In such system, there is often one GPP that functions as a host processor and one or more FPGA-based kernels that function as co-processors to speed-up the processing of special kernels of the application running on the host. FPGA-based accelerators are used in many application domains such as image processing [1], autovision driver assistance [2], bioinformatics applications [3], etc.

While many different interconnect techniques have been developing for communication between processing cores, most FPGA-based accelerator systems use a bus for the interconnect between the host and the kernels (the processing cores). With the rising number of cores, communication between cores increases the requirements on interconnect parameters such as low-latency and area-efficiency. Although bus systems have their advantages [4], they become inefficient when the number of cores rises. On the other hand, Networks on Chips (NoCs) [5] have been proposed as an efficient communication infrastructure in large systems to allow parallel communication and to increase the scalability. However, the major drawback of NoCs compared to bus systems is the increased delay and implementation costs [4].

In this work, we introduce an automated strategy using a communication profiling to design a custom interconnect of kernels in an FPGA-based accelerator system. The main purpose of our work is to improve the communication behavior of the kernels in an existing accelerator system while keeping the amount of hardware (HW) resource usage for the interconnect as low as possible. In state-of-the-art approaches in the literature, input data required for kernel computation is fetched to its local memory (cache) when the kernel is invoked, which delays the start-up of kernel calculations until the data is available. In contrast to these approaches, our approach uses communication profiling to create a custom interconnect for the kernels. The interconnect, then, helps deliver data from one kernel to the others as soon as possible, thereby hiding the data communication time needed for the kernel. The ultimate goal is to have a tailored interconnect infrastructure which is dynamically configured. A custom interconnect design algorithm and an adaptive mapping function using a quantitative data communication profiling of application are proposed to build the interconnect. Our results using four experimental applications show that the proposed system achieves a speed-up of up to $2.87\times$ compared to a baseline system (a conventional FPGA bus-based accelerator system). We also managed to save up to 66.5% energy consumption.

The main contributions of the paper can be summarized as follows: (1) introducing an efficient communication model for FPGA-based accelerator kernels; (2) proposing an automated design algorithm to define a custom interconnect for each specific application with low HW resource usage using data communication profiling.

The rest of this paper is organized as follows: Section II gives a summary of FPGA-based accelerator systems in the literature as well as state-of-the-art data communication optimization techniques. Section III gives an overview of the proposed custom interconnect. Section IV presents in detail the mathematical modeling of system components and our proposed design strategy. In Section V, experimental results show the benefit of the custom interconnect architecture using four experimental applications. Finally, Section VI concludes the paper.

II. State-of-the-art

This section analyzes FPGA-based accelerator systems in the literature based on their interconnect. State-of-the-art data communication optimization techniques for such systems are summarized also in this section.

A. State-of-the-art HW accelerator systems

In recent years, many FPGA-based accelerator systems have been proposed for general purpose computing as well as for specific applications (domains). Figure 1(a) presents a generic architecture of an FPGA-based accelerator system. In such system, the host processor can be a general high-performance CPU (e.g., x86 Intel CPU) or a hardwired embedded processor (e.g., Xilinx PowerPC) or a soft processor (e.g., MicroBlaze or Nios). The kernels are implemented in the reconfigurable area. While the host processor has the main memory to contain application data, the kernels have their local memories to store local data (data cache). These local memories help improving the parallelism between cores. The core communicates with the host, other cores and I/Os through a communication infrastructure.

The following review classifies HW accelerator systems presented in the literature into four different groups based on their communication infrastructure (the interconnect).

1) Bus-based interconnect: Molen [6], Warp processor [7], LegUp [8], IMORC [9] and the experimental systems in [10], [11], [12] use only a bus as the communication infrastructure. The host delivers data input to the kernel when it is invoked and collects the result when the kernel finishes computation through the bus. Other modules such as I/O, DMA, interrupt controller, etc. are also connected with the bus.

2) NoC-based interconnect: MORPHEUS system [13], [14] uses an NoC for data communication of kernels and memory modules. In the MORPHEUS architecture, the control infrastructure is done via an AMBA AHB bus while an exotic Spidergon NoC is used to transfer data among the kernels, the main memory, and the off-chip memory. Although the platform shows very good simulation results, the NoC takes a huge resource toll up to 944,000 ASIC gates. The experimental system in [15], [16] uses an NoC to connect the kernels and the memory modules together. A CoRAM element in each kernel collects data input from the memory modules and sends data output back to them through the NoC. The P2012 architecture [17] uses an asynchronous NoC for data communication among the kernels while communication between the host and the kernels is done by direct memory access (DMA).

3) Shared memory: Convey [18] uses a hybrid-core globally shared memory for data communication of the host and the kernels as well as among the kernels. In [19], Garcia et al. proposed a scalable memory interface for a multicore reconfigurable computing system in which the kernels and the CPU cores communicate through a shared memory hierarchicy. A controller is responsible for issuing memory requests, translating the virtual memory address and guaranteeing the cache-coherence. Shared memory mechanism is used also in [20] through a remote memory access infrastructure. In this system, the CPU and the kernels, connected together via an NoC, have their own local memories. However, through Global Address Space cores (GAScores) (each kernel has a GAScore), the CPU and the kernels can access data stored in the local memories of other kernels.

4) Crossbar: The research in [21] proposed a framework for accelerating large graph problems. The experimental system includes graph processing elements (GPEs) connected with memory modules through a full crossbar. The crossbar contains three different components: FIFOs, an arbiter and multiplexer. The round-robin algorithm is used to schedule communication between the GPEs and the memory.

While the prototype version of [6], [7], [8], [10], [11], [12], [16], [20] implements the host and the kernels in the same chip (embedded hardwired or soft processor as the host), the implementation of [9], [14], [17], [18], [21] uses different chips for the host and the kernels. The research in [19] has not had any prototype version yet.

In most FPGA-based accelerator systems presented above, data input required for computation of the kernels is loaded to the corresponding local memory when the kernel is started, while the result is sent back to the host after the kernel computation finishes.

B. State-of-the-art data communication optimization techniques for FPGA-based accelerator systems

Data communication of an FPGA-based accelerator system can be optimized at two different levels: software (SW) and HW. The following sections present the techniques in the literature to optimize data communication of an FPGA-based accelerator system at those levels.

1) Software optimization: SW optimization is a generic approach and can be applied for an existing FPGA-based accelerator system without any HW modification. However, SW optimization depends on the communication behavior of the running application. Additionally, some SW optimizations require specific modules/functions supported by HW platform.

Transferring data input/output between the host and the local memory of the kernel in parallel with the kernel computation was one of the optimization for data communication reported in [1]. Research in [10] developed an operating system service to establish a direct memory map to the address space of kernels and to enable arrays of data to be copied in a single access. However, this approach also needs to transfer data input from the main memory to the local memory of the accelerator through the system bus when the accelerator is invoked. Curreri et al. [22] proposed a visualization tool to detect data communication bottleneck in an FPGA-based accelerator system using DMA in order to make a decision on increasing or decreasing the DMA block size and bandwidth to improve the system performance. Pouchet et al. [23] proposed a framework to optimize data reuse for a class of programs. Because of the data re-utilization, data communication overhead can be reduced.

2) Hardware optimization: Standard interconnect techniques such as point-to-point, bus and NoC can be used as the communication infrastructure in FPGA-based accelerator systems. However, they have their own limitations such as latencies in bus, routing problem in point-to-point or high area overhead in NoC.

In recent years, some efficient interconnect architectures dedicated for FPGAs have been proposed such as DESA NoC [24] or low-cost and specific-application crossbar in [25], [26]. However, they are not targeting HW accelerator system. Meanwhile, other interconnect techniques have been proposed to accelerate the communication behavior of the kernels in an FPGA-based accelerator system because data communication is usually a primary anticipated bottleneck for system performance [27]. Research in [28] proposed a multi-ported cached design for shared memory communication of multiple accelerator kernels. However, this proposal is system-dependency since they assume that the on-chip memory can work at 2× the system clock (clock for kernels). Another interesting work is the CoRAM architecture [29]. In this work, CoRAM modules are used to efficiently collect data input required for kernel computation and to deliver the result back to the memory. A SW control task manages the execution of the CoRAM modules and informs the kernels when data input is ready. In the IMORC architecture [9], asynchronous FIFOs are inserted into the communication channels between the cores and the memories to provide a sufficient bandwidth as well as to help improving the system performance by decoupling core execution and memory accessing.

However, all the above approaches have not taken the actual data communication pattern between the kernels into consideration yet. Moreover, they are proposed for specific systems rather than for a generic one. Different from those approaches, our work uses data communication profiling information of each application/domain to define an efficient custom interconnect for the kernels. Additionally, our generic approach can be used for an existing FPGA-based accelerator system.

III. Overview of Our Approach

This section presents a baseline system that we use to compare with our proposed system and shows an overview of our approach.

A. Baseline system

Similar to most presented FPGA-based accelerator systems in Section II-A, we implement a baseline system that contains one host processor and some FPGA-based accelerator kernels processing some computationally intensive functions of the application. The input data required for kernel computation is fetched to the local memory of the kernel and the result is sent back to the host after the finishing of the kernel. Data is transferred through the communication infrastructure.

Consider a system with n FPGA-based kernels; a kernel i (with $0 < i < n-1$) is defined as in Equation 1

$$HW_i(\tau_i, D^H_{i(in)}, D^K_{i(in)}, D^H_{i(out)}, D^K_{i(out)}) \tag{1}$$

in which

- τ_i is the computation time of the kernel;
- $D^H_{i(in)}$ and $D^K_{i(in)}$ are the total amount of input data for HW_i generated by functions in the host and by other kernels, respectively;
- $D^H_{i(out)}$ and $D^K_{i(out)}$ are the total amount of output data of HW_i consumed by functions in the host and by other kernels, respectively.

Please note that, all input data ($D_{i(in)} = D^H_{i(in)} + D^K_{i(in)}$) required for kernel computation is fetched from the host and all output result ($D_{i(out)} = D^H_{i(out)} + D^K_{i(out)}$) is sent back to the host following the presented model. However, we distinguish data of the host ($D^H_{(in/out)}$) and of the other kernels ($D^K_{(in/out)}$) to do a comparison with our proposed model later.

The total execution time of n kernels is shown in Equation 2 in which θ is the average time for transferring one byte of data through the communication infrastructure. In this equation, $\sum_{i=0}^{n-1} \tau_i$ is the total computation time while $\sum_{i=0}^{n-1} (D_{i(in)} + D_{i(out)})\theta$ is the total communication time. While the computation time depends on the kernels, the communication time depends on the data movement.

$$T_b = \sum_{i=0}^{n-1} \tau_i + \sum_{i=0}^{n-1} (D_{i(in)} + D_{i(out)})\theta \tag{2}$$

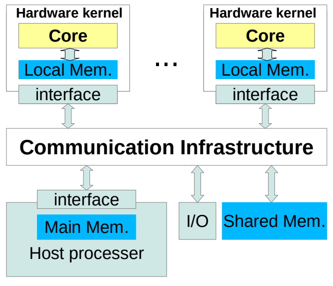
Although the fetching phase can be done in pipeline with the computation phase, this mechanism depends on the actual behavior of the application. Therefore, we use a general model that is compatible with most applications (domains) as a baseline system.

B. Data communication profiling tool

In this work, we use the QUAD toolset [30] as the data communication profiling tool. The toolset provides a comprehensive overview of the data communication behavior of an application. QUAD measures the actual data transferred between producer function and consumer function. The exact amount of byte transfers and the number of Unique Memory Addresses (UMAs) used in the transfer process are also measured. The output of QUAD is data communication profiling information shown as a graph in which the amount of data transfer among functions is shown. Based on this measurement, we can recognize which data communication should be and can be optimized for achieving speed-up.

C. Our approach

An important challenge in FPGA-based accelerator system is to get the data to the computing core that needs it. The goal is of course to hide the communication delay such that a performance improvement can be observed. The resource allocation decision requires detailed and accurate information on the amount of data that is needed as input and what will be produced as output. Evidently, there exists dependencies between computations and data produced by one kernel will be needed by another kernel. In order to have an efficient allocation scheme where the communication delays can be hidden as much as possible, a detailed profile of the data communication patterns is necessary for which then the most appropriate interconnect infrastructure can be generated. Such communication patterns can be specific for each application (domains) and could therefore lead to different interconnects.



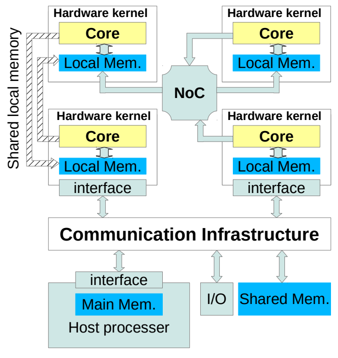


Fig. 1. (a) The generic HW accelerator architecture; (b) The generic HW accelerator system with our custom interconnect for the communication of kernels

Using the detailed profile of the data communication patterns, a kernel knows exactly which kernels will consume its output. Therefore, the kernel can deliver its output ($D^K_{i(out)}$) directly to the consuming kernels when the output is available. This approach is different from the approaches presented in Section II which collect data for a kernel whenever it is invoked. To support this model in a conventional FPGA-based accelerator system, beside the existing communication infrastructure which is usually used for data communication between the host and the kernels, a custom interconnect for the kernels is implemented. The custom interconnect, then, helps the kernel deliver its output. The custom interconnect in our work includes an NoC, shared local memory mechanism or both. We share the local memories of two kernels that only communicate together while NoC is used for a group of communicating kernels (more than two kernels). Figure 1(b) depicts our concept system in which a custom interconnect is implemented to improve data communication of kernels in a generic FPGA-based accelerator system (Figure 1(a)). Using only the NoC as the custom interconnect is an alternative solution. However, the more kernels connect to the NoC, the more routers are needed. This, in turn, increases the size of the custom interconnect. Another solution is to use only shared memory mechanism as the interconnect as proposed in [28]. However, when the number of kernels increases, the overhead for competition to access the shared memory is increased also.

IV. Automated Interconnect Design

In this section, the design strategy using the data communication profiling is introduced in order to define a custom interconnect of kernels for an existing FPGA-based accelerator system with optimized execution time and resource usage. To derive a mathematical model for performance estimation, we denote communication between two kernels as $[HW_i \rightarrow HW_j : D_{ij}]$ in which HW_i sends D_{ij} byte to HW_j. This communication behavior can be extracted from data communication profiling of the application.

A. Modeling system components

1) Modeling shared local memory: In this model, we consider to share the local memories of two kernels in which one kernel (HW_i) sends its output $D^K_{i(out)}$ to another kernel (HW_j) only and HW_j receives input data $D^K_{j(in)}$ from HW_i only (i.e., $[HW_i \rightarrow HW_j : D_{ij}]$ and $D^K_{i(out)} = D^K_{j(in)} = D_{ij}$). With the shared local memory, D_{ij} byte of data can be used without any transferring. Hence, compared to the baseline model, the communication time for this data segment is reduced by $\Delta_c = 2D_{ij}\theta$ (one time from the local memory of HW_i to the host and one time from the host to the local memory of HW_j).

When implemented on FPGAs, most accelerator systems use block RAM (BRAM) as the local memory. BRAM in modern FPGA usually has two ports. Therefore, in a general case, we use a crossbar to share the local memories of two communicating kernels because one port is usually used for the host communication ($D^H_{(in/out)}$) (the same situation is reported in [28]). The crossbar switches data from the cores to the corresponding local memory based on the address of data. The crossbar does not introduce any communication overhead because it does not change the structure of data. In other words, we do not need to encode and decode data format. In a special case in which HW_j does not communicate with the host and other modules connected to the system communication infrastructure, i.e., $D^H_{j(in)} = D^H_{j(out)} = 0$, HW_i and HW_j can share the local memory without the crossbar. Figure 2 illustrates the shared local memories solution with the crossbar (*kernel* 1 and *kernel* 2) and without the crossbar (*kernel* 3 and *kernel* 4).

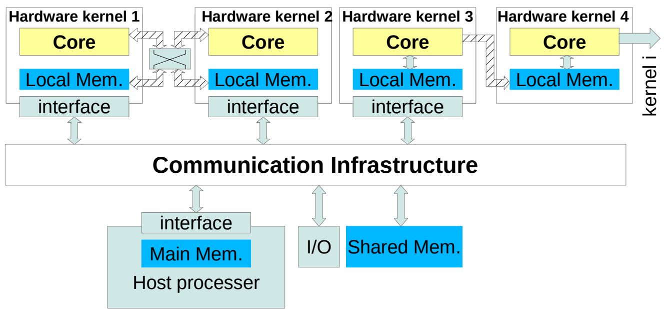


Fig. 2. Shared local memories with and without crossbar in an FPGA-based accelerator system

2) Modeling NoC-based interconnect: NoCs are an established and widely used as interconnect mechanism providing parallelism and high performance. In this model, we use the NoC as the interconnect of a group of kernels. The NoC is used to transfer data from one kernel to the local memories of other kernels. Figure 3 shows a group of kernels using an NoC as their interconnect. An alternative solution is using only the NoC as interconnect of the whole system, i.e., the communication infrastructure part which is used for data communication between the host, the shared memory modules, the I/O modules and the kernels in Figure 3 is eliminated. However, this solution will incur a higher HW overhead for the network adapters at the host and the I/O. Moreover, most

FPGA-based accelerator systems have a predefined communication infrastructure for these components (the host, the shared memory, the I/O, etc.). Additionally, in some HW accelerator systems (such as the Convey architecture [18]), the communication infrastructure of these systems is not reconfigurable. Therefore, ddding an NoC to accelerate the communication behavior of the kernels in an FPGA-based accelerator system is more suitable than modifying the whole system.

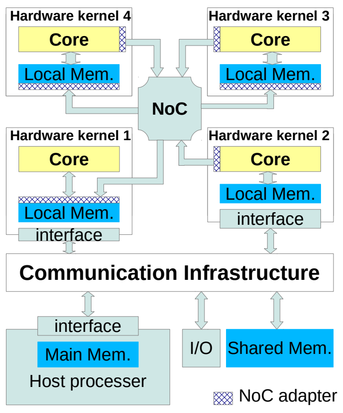


Fig. 3. The NoC is used as interconnect of the kernels in an FPGA-based accelerator system

With the NoC and data communication profiling information, data communication of the kernels is done in parallel with their execution. In other words, the output of one kernel is sent directly to the local memories of the consuming kernels through the NoC rather than stored in its local memory. Hence, kernel i does not need to collect data input $D^K_{i(in)}$ from the host and send data output $D^K_{i(out)}$ back to the host. Consequently, the communication time of the kernels is hidden. Compared to the baseline model, the NoC reduces the execution time by $\Delta_n = \sum_{i=0}^{n-1}(D^K_{i(in)} + D^K_{i(out)})\theta$.

In this model, kernels and local memories are connected to NoC routers through adapters. We consider a general NoC in which each router supports either one kernel or one local memory. Therefore, the number of routers is a sum of the components connected to the NoC. The more routers are used, the larger HW resources are required. That is the reason why we consider the shared memory solution to have an optimized resource usage.

Additionally for further optimization on HW resource usage, based on the communication topology of each specific application, we define a different connection topology of the kernels and the local memories to the NoC (not all the kernels and the local memories which are not applied the shared local memory solution are connected to the NoC). For example, in Figure 3, *kernel 1* and local memory in *kernel 2* are not connected to the NoC because we assume that $D^K_{1(out)} = 0$ and $D^K_{2(in)} = 0$. A detail of this adaptive mapping is presented in Section IV-B.

3) Modeling parallel processing: One of the main advantage of the NoC is that it ensures the parallelism of processing elements connected to it. In some specific applications, especially in multimedia applications such as image or video processing, data can be processed as streaming input [1]. Using these concepts, the parallel processing can be applied in three different cases to further optimize the system performance.

- **Case 1**: pipelining data communication between the kernels and the host. Assume that the input data from the host $D^H_{i(in)}$ of kernel HW_i is segmented into two segments S_1 and S_2. The host fetchs S_1 to the local memory of HW_i. HW_i then processes S_1 while the host fetches S_2. The same way is applied for the output data $D^H_{i(out)}$. Compared to the NoC model, the execution time is reduced by $\Delta_{p1} = min(\frac{D^H_{i(in)}}{2}\theta, \frac{\tau_i}{2}) + min(\frac{D^H_{i(out)}}{2}\theta, \frac{\tau_i}{2}) - O$, where O is the overhead in streaming processing.
- **Case 2**: pipelining the processing of kernels. Assume kernels HW_j can process the result of kernel HW_i in streaming and the data is segmented into S_1 and S_2. HW_i processes S_1 first. HW_i, then, processes S_2 while HW_j processes the first segment. Compared to the NoC model, the execution time is reduced by $\Delta_{p2} = min(\frac{H_i}{2}, \frac{H_j}{2}) - O$, where O is the overhead in streaming processing.
- **Case 3**: duplicating computationally intensive kernels. If a time consuming kernel can be executed in parallel with different data input segments with the overhead O, it can be duplicated to reduce the execution time. Assume kernel HW_i is duplicated, compared to the NoC model, the execution time is reduced by $\Delta_{dp} = \frac{H_i}{2} - O$, where O is the overhead for parallel processing.

B. Design strategy

In this section, an automated design strategy is proposed to define an efficient custom interconnect for a specific application in terms of optimized communication time and low HW resource usage. Evidently, FPGA-based kernels and their communication behavior are different from one application to the others. Therefore, a specific application should have a specific custom interconnect to efficiently get data to the kernels that need it. Algorithm 1 shows the pseudo code of the custom interconnect design algorithm. The result of the algorithm is an interconnect with the most optimized communication time and HW resource usage using the presented modeling system components. The algorithm, first, selects functions which are the most computationally intensive and suitable for accelerating on FPGA (i.e., those functions that can be implemented in HW) (line 1). The most computationally intensive functions are considered for HW duplication if acceptable (i.e., if the most computationally intensive functions can be parallelized and HW resource is available) (line 2-6). The algorithm, then, uses the QUAD open source toolset [30] to generate the quantitative data communication profiling of the application (line 7). Based on this profiling, an efficient custom interconnect using the presented solutions is built.

In this algorithm, the shared local memory solution (line 8-13) is investigated first as explained in Section IV-A1. This communication can also be done by the NoC. However, with the NoC, we need four routers (two for kernels and two for their local memories). Keeping in mind that the HW resources usage for four routers is 5× larger than the HW resources usage for shared local memory solution (using a crossbar or directly sharing the local memory). The amount of HW resources usage is shown in Section V-B for the comparison. The shared local memory solution represents the optimal solution compared to the NoC in term of HW resource usage. Therefore, this solution is considered before the NoC solution.

Algorithm 1 Custom interconnect design

```
Input: Application source code
Output: The most optimized interconnect
1:  $L_{hw} \leftarrow$ List of the most computationally intensive functions suitable to implement on HW;
2:  for each $HW$ in $L_{hw}$ do
3:      if $HW$ satisfies the duplication solution ($\Delta_{dp} > 0$) & resource is available then
4:          Duplicate $HW$ in $L_{hw}$
5:      end if
6:  end for
7:  $G \leftarrow$ Quantitative data communication profiling for functions in $L_{hw}$;
8:  for each communication $[HW_i \rightarrow HW_j : D_{ij}]$ in $G$ do
9:      if $D^K_{i(out)} = D^K_{j(in)} = D_{ij}$ then
10:         Apply the shared local memory solution for $HW_i$ and $HW_j$
11:         Remove $HW_i$ from $L_{hw}$
12:     end if
13: end for
14: Map all $HW$ in $L_{hw}$ to the NoC using adaptive mapping
15: Check the parallel solution (Case 1 & 2) for all $HW$
```

The next step is to map the remaining kernels which are not connected using the shared local memory solution to the NoC (line 14). As explained in the Section IV-A2, an efficient mapping method to map the kernels and the local memories to the NoC and the communication infrastructure reduces the HW resource usage while keeping the communication time in minimized. Therefore, we propose an adaptive mapping function to map the remaining kernels and their local memories to the NoC and the system communication infrastructure. Finally, the parallel processing solution is considered to further reduce the execution time if acceptable (line 15).

The proposed adaptive mapping function is shown in Equation 3.

$$f : Communication \rightarrow Interconnect \tag{3}$$

where the $Communication$ and the $Interconnect$ values are defined below.

There are three different cases in which a kernel receives data input:

1) from other kernels only (R_1);
2) from the host only (R_2);
3) from both other kernels and the host (R_3).

Similarly, there are three different cases in which a kernel sends data output:

1) to other kernels only (S_1);
2) to the host only (S_2);
3) to both other kernels and the host (S_3).

In total, for each HW accelerator, there are nine different data communication topology cases as in Equation 4.

$$Communication = \{R_1, R_2, R_3\} \times \{S_1, S_2, S_3\} \tag{4}$$

There are two options for a connection between a kernel and the NoC

1) the kernel is not connected with the NoC (K_1);
2) the kernel is connected with the NoC (K_2).

Similarly, there are three options for a connection of a local memory with the system communication infrastructure and the NoC

1) the local memory is connected to the communication infrastructure only (M_1);
2) the local memory is connected to the NoC only (M_2);
3) the local memory is connected to both (M_3).

In total, for each kernel and its local memory, there are six different interconnect topology cases as in Equation 5.

$$Interconnect = \{K_1, K_2\} \times \{M_1, M_2, M_3\} \tag{5}$$

Table I shows the mapping of the communication topology to the interconnect topology. The interconnect value $\{K_1, M_2\}$ (the kernel is not connected to the NoC while its local memory is connected to the NoC only) is not feasible as the result of the HW accelerator will be inaccessible by any other function.

TABLE I
ADAPTIVE MAPPING FUNCTION

Communication	Interconnect
$\{R_1, S_1\}$	$\{K_2, M_2\}$
$\{R_1, S_2\}, \{R_3, S_2\}$	$\{K_1, M_3\}$
$\{R_1, S_3\}, \{R_3, S_1\}, \{R_3, S_3\}$	$\{K_2, M_3\}$
$\{R_2, S_1\}, \{R_2, S_3\}$	$\{K_2, M_1\}$
$\{R_2, S_2\}$	$\{K_1, M_1\}$

To reduce the NoC latency, a kernel and its communicating local memories should be mapped to the NoC routers in such a way that the distance of these routers is shortest. For instance, if a kernel is mapped to a router at the coordination (x, y) then the ideal location for the local memory to which it communicates is either $(x - 1, y)$, $(x + 1, y)$, $(x, y - 1)$, or $(x, y + 1)$.

The objective of this mapping is to define the most optimized mapping in terms of HW resource usage. An alternative simpler solution is to map all the kernels and all their local memories to both the NoC and the system communication infrastructure. However, this mapping solution requires the maximum number of routers as well as network adapters. Different from other state-of-the-art mapping algorithms for an FPGA NoC-based system such as [31], [32] which map application tasks to NoC only, our work considers to map both the kernels and the memory to both the NoC and the system communication infrastructure of the FPGA-based accelerator system.

V. Experimental Results

In this section, we present our experimental results with four different applications in both the baseline system and our proposed system.

A. Baseline system setup

In order to validate our custom interconnect design as well as show our advantages, we first implement the baseline system as presented in Section III-A. A Xilinx ML510 [33] board containing an xc5vfx130t FPGA device is used for our experiments. The PowerPC - an embedded processor of the FPGA device is used as the host (run at 400MHz), and the kernels (run at 100MHz) are mapped onto the reconfigurable area. The Xilinx PLB bus is used as the communication infrastructure. The main memory for the host is the off-chip SDRAM connected to the PowerPC. We use BRAM as the local memories of the kernels. Other I/O modules such as UART, Flash Memory Controller, Timer, Interrupt, etc., are also connected with the system through the communication infrastructure. The execution of the baseline system follows the model presented in Section III-A.

Four applications are used for the following experiments. Those are the Canny edge detection application [34], *jpeg* decoder application [35], KLT feature tracker [36] and Fluid simulation [37]. The HW kernels for each application are generated automatically by the DWARV compiler [38]. The systems are synthesized with Xilinx ISE 13.2 without any manual optimization.

To verify the acceleration ability of the baseline system, we first run the experimental applications on the PowerPC (at 400MHz) only to extract the SW execution time. Please note that most FPGA-based accelerator systems presented in Section II-A compare their systems to a host processor (SW time) running at a low frequency (e.g. 85MHz in [7], 75MHz in [8], 125MHz in [10], 100MHz in [11], etc.) while our comparison is done against the host running at 400MHz. We run the applications with the baseline system and get computation time and communication time of kernels and overall application. Figure 4 shows the speed-up of the baseline system compared to the SW as well as ratio between communication time and computation time in the baseline system. The baseline system achieves speed-ups of up to 4.23× for the kernels and 2.93× for the overall application. However, the performance of the baseline system is slower than the SW in case of the *jpeg* application because the large communication time (we measured that communication time is 3.63× larger than computation time). The same situation is also reported in [10] even when their comparison is done against a SW running on a host at 125MHz. Although the baseline system achieves a speed-up of 1.62× for the overall application and of 1.98× for the kernels compared to the SW in average, communication time of the kernels is larger than computation time (the ratio is about 2.09×). Therefore, reducing data communication time leads to a significant improvement in system performance.

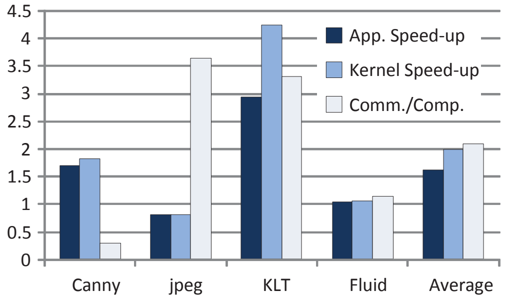


Fig. 4. The speed-up of the baseline system compared to the SW and the ratio between communication and computation time

B. Proposed system setup

To do experiment with the proposed custom interconnect, beside the Xilinx PLB bus used as the system communication infrastructure we developed a 2 × 2 crossbar for the shared local memory solution and adapted the NoC presented in [39] into our system. The NoC adapters for both the kernels and the local memories are also developed. Table II shows the HW resource usage and maximum frequency for HW modules used as interconnect. Please keep in mind that the crossbar is not always used in the shared local memory solution as explained in Section IV-A1. The crossbar does not introduce any communication overhead because data does not need to be encoded and decoded when transferred through the crossbar.

TABLE II
HW resource utilization (#LUTs/#Registers) and frequency of interconnect components

Component	Resource	Max. frequency
Bus	1048/188	345.8MHz
Crossbar	201/200	N/A
NoC Router	309/353	150MHz
NA HW Accelerator	396/426	422.5MHz
NA local memory	60/114	874.2MHz

NA: Network adapter for the communication between the HW module and the NoC

The proposed system is implemented at the same frequency with the baseline system. Here, we present the details of the *jpeg* decoder from the PowerStone benchmark. The four functions which are most computationally intensive and suitable for HW implementation are chosen to accelerate by the HW kernels ($L_{hw} = \{$*huff_dc_dec*, *huff_ac_dec*, *dquantz_lum*, *j_rev_dct*$\}$ - Line 1 in Algorithm 1). The most computationally intensive function *huff_ac_dec* is chosen to duplicate, according to Line 3-4 in Algorithm 1. Other functions are executed on the host (the PowerPC in this experiment). The application is analyzed to extract the data communication profiling information (depicted as a graph in Figure 5) by the QUAD tool.

According to the graph, *dquantz_lum* sends data to *j_rev_dct* only while *j_rev_dct* consumes data generated by the host and

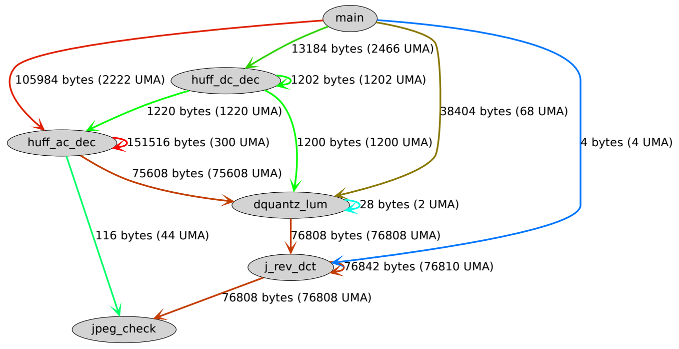


Fig. 5. Data communication profiling for the *jpeg* decoder

dquantz_lum; the shared local memory solution is applied to these kernels (Line 9-10 in Algorithm 1). The NoC is used for the interconnect of *huff_dc_dec*, two kernels of *huff_ac_dec* and *dquantz_lum* (Line 14 in Algorithm 1). As shown in the graph, *huff_dc_dec* consumes data from the host only (case R_2) and sends data to other kernels only (case S_1), the corresponding kernel for this function is connected to the NoC while its local memory is connected to the system communication infrastructure (the PLB bus in this case) ($\{R_2, S_1\} \rightarrow \{K_2, M_1\}$). The connections of the kernels of other functions are deduced in a similar way. All the resulting connections are shown in the Figure 6.

As stated above, we performed our experiment with the BRAM as the local memories of the kernels. Each BRAM has only two ports while local memories of two *huff_ac_dec* kernels (*kernel* 1 and *kernel* 2 in Figure 6) are accessed by three different components (the host, the NoC adapter and the kernel core). Therefore, a multiplexer is used.

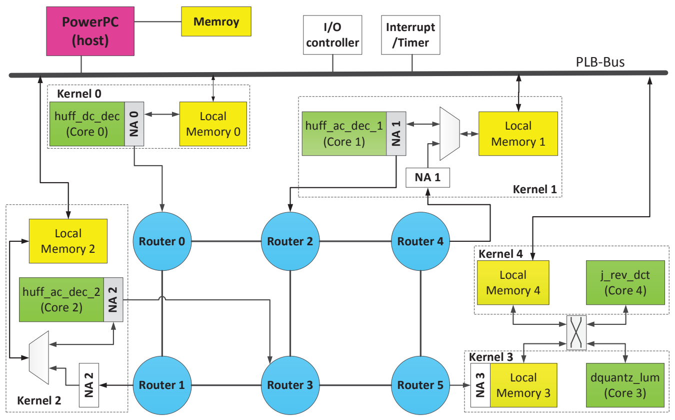


Fig. 6. The proposed system for the *jpeg* decoder application

C. Comparison and Discussion

Table III shows the speed-up of the overall application (column 2 and 4) and of the kernels (column 3 and 5) of the proposed system with respect to both SW and the baseline system. As shown in the table, when the proposed custom interconnect is exploited, it achieves a speed-up of the overall application and of the kernels by up to 3.72× and 6.58× when compared to SW, respectively. Figure 7 compares the speed-up of the proposed system with both SW and baseline. The first two chart bars illustrate the speed-ups of the overall application and of the kernels with respect to SW while the last two chart bars show speed-ups with respect to the baseline system. Compared to the baseline system, speed-ups of up to 3.08× for the kernels and 2.87× for the overall application are obtained (both in the case of the *jpeg* application).

TABLE III
SPEED-UP OF THE PROPOSED SYSTEM WITH RESPECT TO SW AND THE BASELINE SYSTEM

App.	**w.r.t Software**		**w.r.t Baseline**	
	Application	Kernels	Application	Kernels
Canny	3.15×	3.88×	1.83×	2.12×
jpeg	2.33×	2.5×	2.87×	3.08×
KLT	3.72×	6.58×	1.26×	1.55×
Fluid	1.66×	1.68×	1.59×	1.60×

App.: Application

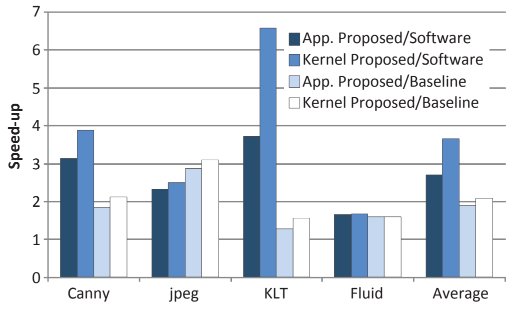


Fig. 7. The overall application and the kernels speed-up of the baseline and our systems

Table IV presents the HW resource utilization for the whole system of the baseline, our system and the NoC-only system, in terms of the number of FPGA look-up tables (LUTs) and the number of FPGA registers. The NoC-only system is a system in which the parallel solution is applied, but only NoC is used for the interconnect of kernels (shared local memory solution is not used). Our proposed communication model in which data is sent through the NoC from the producing core directly to the consuming core as soon as possible is used. However, our adaptive mapping algorithm is also not applied in this system. As shown in the table, our system saves up to 33.1% LUTs and 30.2% Registers compared to the NoC-only system. This result validates our goal which is to optimize the communication time while keeping the minimized resources usage of the interconnect. Without our strategy, the system is either only bus-based or only NoC-based. The bus-based system (baseline) is a low performance system while the NoC-only system uses more HW resources than our system. Meanwhile, our system (both the shared memory solution and the NoC are used as interconnect between kernels) achieves the same performance and uses less resources than the NoC-only system. Figure 8 presents the comparison of resources used for interconnect and for the kernels in our system normalized to the resources used for computing. The interconnect uses only 40.7% resources compared to the resources used for computing at most.

TABLE IV
HW RESOURCE UTILIZATION (#LUTS/#REGISTERS)

App.	Baseline	Our system	NoC only	Solution
Canny	9926 12707	15227 18657	17894 21059	NoC, SM, P
jpeg	11755 11910	20837 20900	23180 23188	NoC, SM, P
KLT	4721 5430	4921 5631	7358 8070	SM
Fluid	19125 28793	24156 36100	24552 36110	NoC

App.: Application; SM: Shared memory; P: Parallel

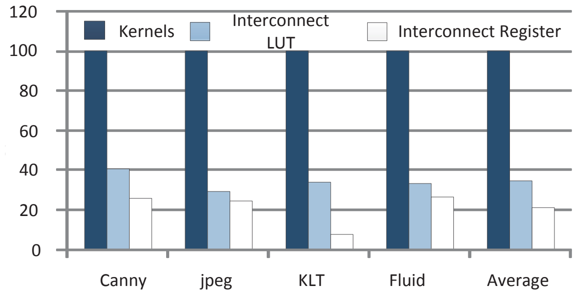


Fig. 8. Interconnect resource usage normalized to the resource usage for the kernels.

Figure 9 presents the comparison of the energy consumption between the baseline system and our system normalized to the energy consumption of the baseline system. We used the Xilinx XPower Analyzer 13.2 tool to estimate the power consumption of each application in the two systems. The energy consumption is given by the product of the power consumption and the execution time. For both systems, the power consumption is almost identical, with a minor increase in our system (due to the increasing of resource usage for the custom interconnect). Therefore, our system consumes less energy consumption per application due to the reduction in execution time. As shown in the figure, our system outperforms the baseline in all applications in terms of energy consumption. The maximum energy saved is 66.5% for the *jpeg* application.

VI. CONCLUSION

In this paper, we presented an automated design strategy to define an efficient custom interconnect for kernels in FPGA-based accelerator systems. The custom interconnect includes an NoC or shared local memories solution (directly sharing or using a crossbar) or both. We also presented an algorithm to automate the design of the custom interconnect based on quantitative data communication profiling of applications. An adaptive mapping to map HW accelerators and their local memories to the interconnect is also introduced. We compared our proposed system with a baseline architecture - an FPGA bus-based accelerator system. The results show that the proposed system achieves an overall application speed-up of 3.72× with respect to SW and of 2.87× compared to the baseline system. Due to the reduced execution time, energy reduction of up to 66.5% was obtained, compared to the baseline system. Runtime reconfigurability is the next step in our work such that each application can dispose of its best interconnect infrastructure leading to faster execution and less overall energy consumption.

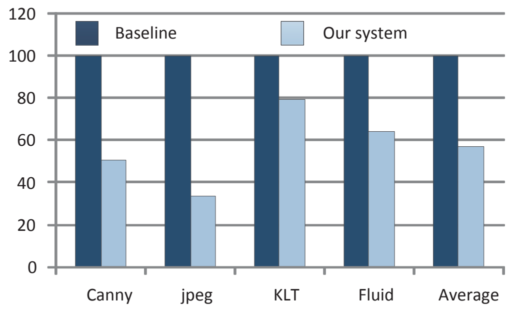


Fig. 9. Energy consumption comparison between the baseline system and our system normalized to the baseline system.

ACKNOWLEDGMENT

This work has been funded by the projects Smecy 100230, iFEST 100203, REFLECT 248976 and Vietnam Ministry of Education and Training.

REFERENCES

[1] J. Cong and Y. Zou, “Fpga-based hardware acceleration of lithographic aerial image simulation,” *Reconfig. Technol. Syst.*, pp. 1–29, 2009.

[2] C. Claus and W. Stechele, “AutoVision-reconfigurable hardware acceleration for video-based driver assistance,” in *Dynamically Reconfigurable Systems*. Springer Netherlands, 2010, pp. 375–394.

[3] S. Sarkar, T. Majumder, A. Kalyanaraman, and P. Pande, “Hardware accelerators for biocomputing: A survey,” in *IEEE International Symposium on Circuits and Systems*, 2010, pp. 3789–3792.

[4] P. Guerrier and A. Greiner, “A generic architecture for on-chip packet-switched interconnections,” in *DATE*, 2000, pp. 250–256.

[5] L. Benini and G. De Micheli, “Networks on chips: a new SoC paradigm,” *Computer*, pp. 70–78, 2002.

[6] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. Panainte, “The MOLEN polymorphic processor,” *Computer*, pp. 1363–1375, 2004.

[7] R. Lysecky and F. Vahid, “Design and implementation of a MicroBlaze-based Warp processor,” *Embedded Computing System*, pp. 1–22, 2009.

[8] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, “Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems,” *Embed. Comput. Syst.*, Sep. 2013.

[9] T. Schumacher, C. Plessl, and M. Platzner, “IMORC: An infrastructure and architecture template for implementing high-performance reconfigurable FPGA accelerators,” *Micropro. & Microsys.*, pp. 110–126, 2012.

[10] A. Ismail and L. Shannon, “FUSE: Front-end user framework for O/S abstraction of hardware accelerators,” in *FCCM*, 2011, pp. 170–177.

[11] C. Pilato, A. Cazzaniga, G. Durelli, A. Otero, D. Sciuto, and M. Santambrogio, “On the automatic integration of hardware accelerators into FPGA-based embedded systems,” in *FPL*, 2012, pp. 607–610.

[12] S. Neuendorffer and F. Martinez-Vallina, “Building Zynq® accelerators with Vivado® high level synthesis,” in *FPGA*, 2013, pp. 1–2.

[13] A. Grasset, P. Millet, P. Bonnot, S. Yehia, W. Putzke-Roeming, F. Campi, A. Rosti, M. Huebner, N. Voros, D. Rossi, H. Sahlbach, and R. Ernst, “The morpheus heterogeneous dynamically reconfigurable platform,” *Parallel Programming*, pp. 328–356, 2011.

[14] N. S. Voros and et al., “Morpheus: A heterogeneous dynamically reconfigurable platform for designing highly complex embedded systems,” *Embed. Comput. Syst.*, pp. 70:1–70:33, Apr. 2013.

[15] E. S. Chung, J. C. Hoe, and K. Mai, “Coram: an in-fabric memory architecture for fpga-based computing,” in *FPGA*, 2011, pp. 97–106.

[16] E. S. Chung, M. K. Papamichael, G. Weisz, J. C. Hoe, and K. Mai, “Prototype and evaluation of the CoRAM memory architecture for FPGA-based computing,” in *FPGA*, 2012, pp. 139–142.

[17] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, “P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator,” in *DATE*, 2012, pp. 983–987.

[18] Convey Computer, “Convey reference manual,” 2012.

[19] P. Garcia and K. Compton, “A scalable memory interface for multicore reconfigurable computing systems,” in *FPT*, 2011, pp. 1–8.

[20] R. Willenberg and P. Chow, “A remote memory access infrastructure for global address space programming models in fpgas,” in *FPGA*, 2013, pp. 211–220.

[21] B. Betkaoui, D. Thomas, W. Luk, and N. Przulj, “A framework for fpga acceleration of large graph problems: Graphlet counting case study,” in *FPT*, 2011, pp. 1–8.

[22] J. Curreri, G. Stitt, and A. George, “Communication visualization for bottleneck detection of high-level synthesis applications,” in *FPGA*, 2012, pp. 33–36.

[23] L.-N. Pouchet, P. Zhang, P. Sadayappan, and J. Cong, “Polyhedral-based data reuse optimization for configurable computing,” in *FPGA*, 2013, pp. 29–38.

[24] A. Roca, J. Flich, and G. Dimitrakopoulos, “DESA: Distributed elastic switch architecture for efficient networks-on-FPGAs,” in *FPL*, 2012, pp. 394–399.

[25] J. Y. Hur, T. Stefanov, S. Wong, and K. Goossens, “Customisation of on-chip network interconnects and experiments in field-programmable gate arrays,” *Computers Digital Techniques*, pp. 59–68, 2012.

[26] S. Murali, L. Benini, and G. De Micheli, “An application-specific design methodology for on-chip crossbar generation,” *Computer-Aided Design of Integrated Circuits and Systems*, pp. 1283–1296, 2007.

[27] S. G. Kavadias, M. G. Katevenis, M. Zampetakis, and D. S. Nikolopoulos, “On-chip communication and synchronization mechanisms with cache-integrated network interfaces,” in *Computing frontiers*, 2010, pp. 217–226.

[28] J. Choi, K. Nam, A. Canis, J. Anderson, S. Brown, and T. Czajkowski, “Impact of cache architecture and interface on performance and area of FPGA-based processor/parallel-accelerator systems,” in *FCCM*, 2012, pp. 17–24.

[29] E. S. Chung, J. C. Hoe, and K. Mai, “CoRAM: an in-fabric memory architecture for FPGA-based computing,” in *FPGA*, 2011, pp. 97–106.

[30] S. A. Ostadzadeh, R. J. Meeuws, C. Galuzzi, and K. Bertels, “QUAD: a memory access pattern analyser,” in *ARC*, 2010, pp. 269–281.

[31] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, “Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms,” *J. Syst. Archit.*, pp. 242–255, Jul. 2010.

[32] H. Yu, Y. Ha, and B. Veeravalli, “Communication-aware application mapping and scheduling for noc-based mpsocs,” in *ISCAS*, 2010, pp. 3232–3235.

[33] Xilinx, “Ml510 reference design,” 2009.

[34] J. Canny, “A computational approach to edge detection,” *Pattern Analysis and Machine Intelligence*, pp. 679 –698, 1986.

[35] J. Scott, L. H. Lee, J. Arends, and B. Moyer, “Designing the low-power M•CORE architecture,” in *Power Driven Microarchitecture*, 1998.

[36] J. Shi and C. Tomasi, “Good Features to Track,” in *Computer Vision and Pattern Recognition*, 1994.

[37] J. Stam, “Real-time fluid dynamics for games,” in *the Game Developer Conference*, 2003.

[38] R. Nane, V. Sima, B. Olivier, R. Meeuws, Y. Yankova, and K. Bertels, “DWARV 2.0: A CoSy-based C-to-VHDL hardware compiler,” in *FPL*, 2012.

[39] J. Heisswolf, R. Koenig, and J. Becker, “A scalable NoC router design providing QoS support using weighted round robin scheduling,” in *ISPAW*, 2012, pp. 625 – 632.