# Heterogeneous Hardware Accelerators with Hybrid Interconnect: an Automated Design Approach

Cuong Pham-Quoc[⋆◇], Imran Ashraf[⋆], Zaid Al-Ars[⋆], Koen Bertels[⋆]
[⋆] Delft University of Technology, Netherlands, Email: {P.PhamQuocCuong,I.Ashraf,Z.Al-Ars,K.L.M.Bertels}@tudelft.nl
[◇] Ho Chi Minh City University of Technology, Email: cuongpham@hcmut.edu.vn

***Abstract*—Although heterogeneous multicore systems are widely used in both academia and industry, system performance of such systems does not scale when increasing the number of processing cores. The main reason is due to the communication overhead which increases greatly with the increasing number of cores. In this paper, we propose an automated design approach to build a heterogeneous hardware accelerator system, one of the main trends in heterogeneous multicore, with hybrid interconnect. Our approach takes communication patterns of an application into account so that data communication of computing cores is optimized while keeping hardware resources usage for the whole system minimal. Experimental results in both an embedded and a high-performance computing platforms show that the design approach improves system performance by up to 1.83× for the embedded platform and by up to 1.53× for the high-performance computing platform. Energy consumption of the embedded platform is reduced by up to 50.3% while energy consumption of kernels in the high-performance platform is reduced by up 54.2%, compared to baseline systems.**

## I. Introduction

In recent years, the need for computation grows, especially when we are entering the big data era. However, CPU frequency growth has slowed over the last few years. Consequently, alternatives should be sought to satisfy this requirement. In the other hand, it is possible to integrate more and more transistor on a single chip. At this time, more than 20 billion transistors are able to be integrated on one chip [1]. However, many challenges need to be addressed when such a large number of transistors integrated on a chip such as thermal emission, power consumption, and memory access bottleneck. Therefore, homogeneous and heterogeneous multicore architectures have been introduced to keep improving system performance and to utilize such the large number of transistors. Compared to homogeneous multicore systems, heterogeneous ones offer more computation power and efficient energy consumption [2] because of the utilization of specialized cores for specific functions.

As one of the main trends in heterogeneous multicore, hardware (HW) accelerator systems have been considered as a main approach to continue performance improvement in the future [3], [4]. In such systems, there is often one general purpose processor (GPP) that acts as a host processor and one or more HW accelerators (kernels) that work as co-processors to accelerate the processing of computationally-intensive functions of an application running on the host. Due to the combination of both GPP and application specific kernels, HW accelerators systems are more energy efficient and have higher performance than GPP while still providing a significant degree of flexibility.

Many HW accelerator systems have been proposed in academia such as Molen [5], MORPHEUS [6], etc., and more and more in industry championed by companies such as Maxeler [7], Convey [8], IBM Power 8 [9], Microsoft Catapult [10], etc. HW accelerator technology has been popular for a while in both embedded and high-performance computing. However, one of the open issues in HW accelerator systems is the interconnect design [11]. In such systems, interconnect that plays an important role is a predefined system backbone upon which data is exchanged between all system components. Although data communication is a primary anticipated bottleneck for system performance, interconnect design is not well addressed in most HW accelerator systems in both academia and industry.

Evidently, accelerator kernels and their communication behavior are different from one application to the other. A specific application should have a specific interconnect dedicated for its communication patterns. The specific interconnect should have optimized performance while keeping HW resources usage minimal. In this paper, we propose an automated design approach that takes data communication patterns of an application into account to implement a HW accelerator system for the application with hybrid interconnect. The hybrid interconnect provides the most appropriate support for the communication patterns inside the application while using HW resources as low as possible.

The main contributions of this paper include:

1) summarizing state-of-the-art HW accelerators and hybrid interconnects in both the literature and industry;
2) proposing an automated design approach to design a HW accelerator system with hybrid interconnect for each specific application;
3) presenting the experimental results in both an embedded and a high-performance computing platforms.

The rest of this paper is organized as follows. Section II summarizes state-of-the-art HW accelerators in both acamedia and industry as well as proposed hybrid interconnect architecture in the literature. Section III presents an overview of our approach and our proposed design flow. The automated approach to design a HW accelerator system with hybrid interconnect is introduced in Section IV. Experimental results using both an embedded and a high-performance computing platforms are showns in Section V. Finally, Section VI concludes the paper.

## II. State-of-the-art

In this section, we present state-of-the-art HW accelerator systems in both academia and industry. We classify the sys-

tems based on their communication infrastructure. We also summarize proposed hybrid interconnect in the literature.

### A. Hardware accelerator systems

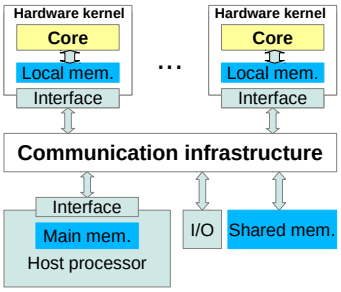


Fig. 1. A generic HW accelerator architecture.

In recent years, many HW accelerator systems have been proposed for general purpose computing as well as for specific applications (domains). Figure 1 shows a generic architecture of a HW accelerator. In such architecture, the host processor can be a general high-performance CPU (e.g., x86 Intel CPU) or an embedded processor (e.g., Xilinx PowerPC) or a soft processor (e.g., MicroBlaze or Nios). The HW accelerator kernels, or HW called kernels, are implemented in HW fabric such as Field Programmable Gate Array (FPGA), Digital Signal Processor (DSP), (Graphic Processor Unit) GPU, etc. While the host processor uses the main memory to store application data, the kernels have their local memories to store local data (buffer) to improve the parallelism between the kernels. A kernel communicates with the host, other kernels and I/Os through a communication infrastructure.

The following review classifies the HW accelerator systems in the literature and industry into four different groups based on the communication infrastructure (the interconnect) of the system: bus, network-on-chip (NoC), shared memory, and crossbar.

- *Bus-based interconnect:* Molen [5], Warp processor [12], IMORC [13], the target systems in [14], [15], [16], and IBM's PowerEN [17] use a bus as the communication infrastructure.
- *NoC-based interconnect:* The MORPHEUS system [6], [18] uses the Spidergon NoC for data communication of kernels and memory modules. The target systems presented in [19], and [20], use a CoRAM element in each kernel to collect data input for the kernel from the memory modules and to send the result back to the memory modules through a NoC. The P2012 architecture [21] uses an asynchronous NoC for communication among the kernels.
- *Shared memory:* Shared memory is used in many commercial HW accelerator systems for high-performance computing. Convey [8], the IBM Power 8 [9], Microsoft Catapult [10], and Intel [22] use shared memory to exchange data between their host processors and kernels or among kernels. Shared memory is also used in [23] through a remote memory access infrastructure.
- *Crossbar:* The research in [24] proposed a framework for accelerating large graph problems using a crossbar for data communication between their graph processing elements (GPEs) and memory modules. The work in [25] used an optimized HW resources and high routability crossbar as interconnect of kernels and memory modules.

### B. Hybrid interconnect architectures

The previous sections classified HW accelerator systems in both the literature and industry into four different groups based on the interconnect architecture. Each interconnect type (bus, shared memory, NoC, crossbar) has its own advantages and disadvantages. While buses are simple and area-efficient, they suffer from low-performance and scalability problems compared to the others because of the serialized communication [26]. A crossbar outperforms a bus in term of system performance because it offers separate paths from sources to destinations [27]. However, it has limited scalability since the area cost increases quadratically when the number of ports increases. While shared local memory can offer an area-efficient solution, its scalability is limited by the finite number of memory ports. Although NoCs have their certain advantages such as high-performance and scalability, they suffer from a high area cost [28]. Therefore, a hybrid interconnect with high-performance, area-efficiency and high scalability is an essential demand.

In recent years, many hybrid interconnect architectures have been proposed. Those architectures can be classified into two categories: mixed topology hybrid interconnect and mixed architecture hybrid interconnect. While different NoC topologies are combined together to form a hybrid interconnect in the first group, the second group includes hybrid interconnects that combine two interconnect types, for example a bus and a NoC. Due to the space limitation, we summarize hybrid interconnects in the second group only. Table I summarizes the proposed mixed architectures hybrid interconnects in the literature. As shown in the table, all the mixed architectures hybrid interconnects combine buses and a NoC to form hybrid interconnects. Beside the static designs, communication rate is usually used as input data to design hybrid interconnects. However, communication rate may change from time to time. None of the above proposed hybrid interconnects takes application quantitative data communication patterns into account.

In contrast to those proposed hybrid interconnects, our hybrid interconnect design takes communication patterns inside an application into consideration. In other words, we use the application quantitative data communication profiling to design a HW accelerator system with a hybrid interconnect.

## III. Design Approach

### A. Overview

In conventional execution models of HW accelerator systems in the literature, data input required for kernel computation is fetched to its local memory when the kernel is invoked and data output is sent back to the main memory when the kernel is finished [16], [40]. This delays the start-up of kernel calculations until the whole data is available. Although there are some specific solutions to improve this communication behavior, those solutions are ad-hoc approaches and have not taken data communication patterns of the application into consideration. In contrast to them, we aim to provide a more

TABLE I
MIXED ARCHITECTURE HYBRID INTERCONNECT SUMMARY

| Proposal | Combined architectures | Input data[a] | Experimental platform | Year |
|---|---|---|---|---|
| dTDMA/NoC [29] | Bus/NoC | Communication rate | Simulation | 2006 |
| MECS [30] | Bus-like/NoC | Static | Simulation | 2006 |
| BENoC [31] | Bus/NoC | Static[b] | Simulation | 2009 |
| Das et al. [32] | Bus/NoC | Static | Simulation | 2009 |
| RAMS [33] | Bus/NoC | Memory access rate | Simulation | 2010 |
| Tsai et al. [34] | Bus/NoC | Communication bandwidth | Simulation | 2010 |
| HNoC [35] | Bus/NoC | Static | Simulation | 2010 |
| Giefers et al. [36] | Bus/NoC/ Barrier | Static | FPGA-based platform | 2010 |
| MORPHEUS [18] | Bus/NoC | Static | ASIC-based platform | 2011 |
| *duo* [37] | Bus-like/NoC | Communication rate | Simulation | 2012 |
| Zhao et al. [38] | Bus/NoC | Static | Simulation | 2012 |
| Todorov et al. [39] | Bus/NoC routers | Bandwidth and Latency constraints | Simulation | 2014 |

[a]Which input data the proposal uses to design the proposed architecture, for example task graph or communication pattern.
[b]*Static* means that the proposal does not use any data from any application/domain.

generic solution and take data communication patterns of the application into account. The hybrid interconnect dedicated for each application helps deliver data from one kernel to the other as soon as possible, thereby hiding the data communication time needed for the kernel.

The main purpose of this work is to develop an automated design approach to implement a specific application on an existing HW accelerator system so that the interconnect of the system is the most appropriate support communication patterns inside the application while HW resource usage for the interconnect is minimal. In such system, computationally-intensive functions of the application are accelerated by dedicated HW kernels running on the HW fabric while the rest functions are executed on the host processor. While the predefine system communication infrastructure carries out data communication between the host and the kernels, data exchanged among the kernels is performed by our hybrid interconnect that consists of a NoC, a crossbar, and directly shared local memory.

We target a generic HW accelerator system in which accelerator kernels can be implemented on any HW fabric such as GPU, FPGA, or ASIC. GPU interconnect is not reconfigurable in current day technology. Therefore, our discussion is mainly based on reconfigurable computing platforms. Moreover, almost all well-known HW accelerator systems are implemented on reconfigurable platforms as presented in Section II-A.

### B. Design Flow

Figure 2 depicts our proposed design flow to implement a specific application on an existing HW accelerator system with hybrid interconnect. Below, we discuss each step in detail.

*1) Profiling:* The application is profiled by the QUAD profiling tool [41] to extract communication patterns inside the application and by *gprof* [42] to identify computationally-intensive functions that should be accelerated. Outputs of those

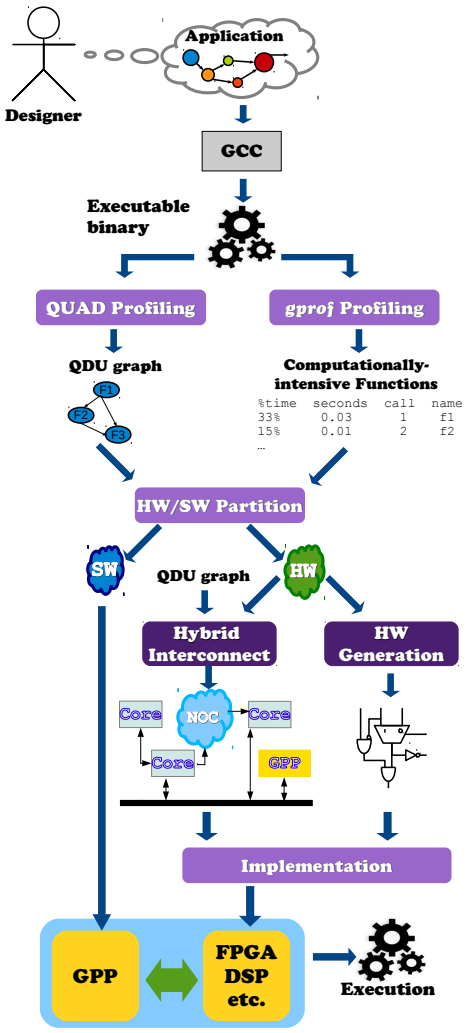


Fig. 2. The proposed design flow.

profiling tools are used in the HW/SW (software) partition step.

*2) HW/SW partition:* Traditionally, the primary objective of HW/SW partition in heterogeneous HW systems is to improve system performance, hence, to gain speed-up. To achieve this goal, the program parts with higher execution time contributions (computationally-intensive functions) are usually mapped onto the HW, while the parts with lower contributions are executed on the host.

*3) Hybrid interconnect:* Using the detailed profile of data communication patterns, an accelerator kernel knows exactly which kernels will consume its output. Therefore, the kernel can deliver its output directly to the consuming kernels when the output is available (instead of transferring its output back to the host as in conventional execution models). To support this model in an existing HW accelerator system, beside the existing communication infrastructure, a hybrid interconnect for the kernels is implemented. Section IV will discuss this hybrid interconnect design in detail.

*4) Hardware generation:* As aforementioned, we use reconfigurable computing platforms as target platforms. High level synthesis tools such as Xilinx Vivado [43], DWARV [44],

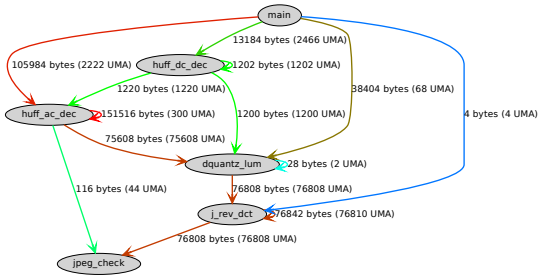


Fig. 3. An example of data communication profiling.

etc., are used to generate synthesizable HW descriptions for accelerated parts from original application source code.

*5) Implementation on target platforms:* Finally, all the design components are synthesized and implemented on target platforms using vendor tools such as Xilinx tools when implementing on Xilinx platforms or Convey PDK toolchain [8] when Convey machines are used. The SW part is executed on the host while the HW part is accelerated on HW fabric. The system communication infrastructure takes care data communication between HW and SW parts while our hybrid interconnect carries out data communication among the kernels.

## IV. Automated Hybrid Interconnect Design

In this section, an automated design strategy using data communication profiling of an application is introduced to build a hybrid interconnect that connects all accelerator kernels of the application. The design strategy results in a system with optimized execution time and resources usage. Data communication profiling of the application can be extracted by the QUAD profiling tool as mentioned in Section III. Figure 3 shows output of the QUAD profiling tool in a graph-based view. Based on this profiling, a kernel knows exactly to where its output should be sent (to the host or to other kernels). To further improve system performance, parallelizing kernel processing is also introduced in this section.

### A. Interconnect components

*1) Shared local memory:* In this work, we consider to share the local memories of two accelerator kernels ($kernel_s$ and $kernel_r$) if the two following conditions are asserted: 1) the two accelerator kernels communicate together; 2) the receiving kernel, $kernel_r$, only processes data generated by the sending kernel, $kernel_s$, or both $kernel_s$ and the host. Two accelerator kernels *dquantz_lum* and *j_rev_dct* in Figure 3 are an example. With the shared local memory, $kernel_r$ can process output of $kernel_s$, called $D_{ij}$ byte, without any data transfer. Due to the fact that the system does not need to transfer $D_{ij}$ byte from local memory to the main memory when $kernel_s$ is finished and copy this data from the main memory to local memory when $kernel_r$ starts, compared to the conventional models, the communication time for this data movement is reduced by $\Delta_c = 2D_{ij}\theta$, where $\theta$ is the average time for transferring one byte of data between the local memory and the main memory.

When implemented on reconfigurable platforms, most accelerator systems use block RAM (BRAM) as the local memory. BRAM in modern FPGA usually has two ports. Therefore, we use a crossbar to share the local memories of two communicating kernels because one port is usually used for the host communication, receiving data input from the host or sending data output to the host (the same situation is reported in [45]). The crossbar does not introduce any communication overhead because it does not change the structure of data. In a special case in which the receiving kernel does not communicate with the host, the two kernels can share their local memories without the crossbar. Figure 4 illustrates the shared local memories solution with the crossbar (*kernel* 1 and *kernel* 2) and without the crossbar (*kernel* 3 and *kernel* 4).

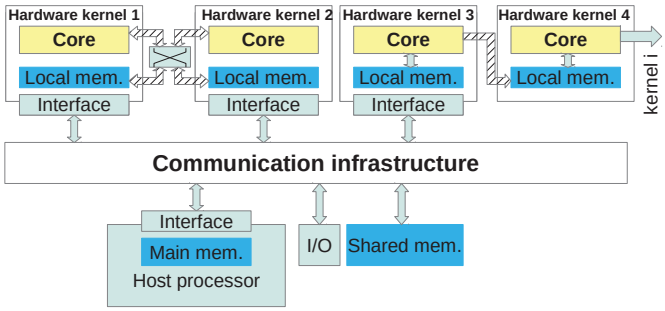


Fig. 4. Shared local memories with and without crossbar in an FPGA-based accelerator system

*2) NoC:* NoCs are an established and widely used as interconnect mechanism providing parallelism and high-performance. In this work, we use a 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 5 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 in Figure 5 is eliminated. However, this solution will incur a higher HW overhead for the network adapters at the host and other components. Moreover, most HW accelerator systems have a predefined communication infrastructure to connect system components (the host, the shared memory, the I/O, etc.). In some HW accelerator systems (such as the Convey architecture [8]), the communication infrastructure is not reconfigurable. Therefore, adding a NoC to accelerate the communication behavior of the kernels in a HW accelerator system is more suitable than modifying the whole system.

With the NoC, 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 rather than stored in its local memory. Hence, a kernel does not need to collect data input produced by other kernels from the main memory and send data output consumed by other kernels back to the main memory as in the conventional models. Compared to the conventional models, the NoC reduces the execution time by $\Delta_n = 2\sum_{i=0}^{n-1} D^K_{i(out)}\theta$, where $n$ is the number of kernels connected to the NoC; $D^K_{i(out)}$ is the amount of data generated by kernel $i$-th and consumed by other kernels.

Additionally, to further optimize HW resources usage, based on the communication patterns of each specific application, we define a connection topology of the kernels and the local

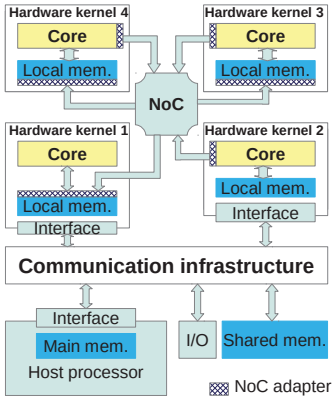


Fig. 5. The NoC is used as interconnect of the kernels in a HW accelerator system

memories to the NoC and the communication infrastructure; i.e., not all the kernels and the local memories which are not applied the shared local memory solution are connected to the NoC. A kernel is connected to the NoC if and only if it sends data output to other kernels. A local memory is connected to the communication infrastructure if the corresponding kernel communicates (sending output or receiving input) with the host while it is connected to the NoC if the kernel receives data input generated by other kernels. For example, in Figure 5, *kernel 1* and local memory of *kernel 2* are not connected to the NoC because we assume that *kernel 1* does not send its output to any other kernels and *kernel 2* receives data input from the host only.

### B. Parallelizing kernel processing

Given the fact that HW accelerator systems have been increasingly used to address parallelizable and data intensive application domains [22] such as image or video processing [40], datacenter services [10], etc., the parallelizing kernel processing can be used to further improve system performance beside the proposed hybrid interconnect. Parallelism can be exploited at two different levels: data parallelism and instruction parallelism.

*1) Data parallelism:* Data parallelism is an execution scenario in which data is partitioned into segments, and concurrent processing kernels process those segments in parallel. In other words, one computationally intensive function can be accelerated by a number of concurrent kernels. Each kernel processes each data segment. Assume that a computationally intensive function has $n$ accelerator kernels, data input for the function is partitioned into $n$ segments. The reduction in processing time of this function compared to one accelerator kernel is $\Delta_{dp} = \frac{\tau_i(n-1)}{n} - O$ where $\tau_i$ is the time for processing the whole data with only one kernel and $O$ is the overhead for data parallelism processing. This overhead depends on the application's algorithm and occurs because extra data needs to be processed to achieve the correct result for each segment [46]. Figure 6 shows a comparison between serial processing and data parallelism processing in which the function is accelerated by three different kernels (Kernel i_1, Kernel i_2, and Kernel_3) and data is partitioned into three segments.

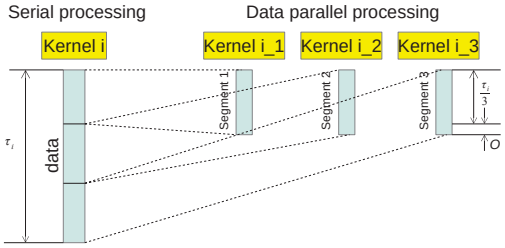


Fig. 6. An example of data parallelism processing compared to serial processing

*2) Instruction parallelism:* Instruction parallelism is an execution scenario in which accelerator kernels of functions form a pipeline to process a stream of data segments. Each kernel of each accelerated function establishes a pipeline stage. Data segments are streamed through those stages. Different from the serial execution scenario, in instruction parallelism, the kernels are working at the same time. The pipeline depth is the number of data segments processed. Assume that there are $n$ processing stages, the total execution time in the serial scenario when the proposed hybrid interconnect takes care of the data communication between the kernels is evaluated by Equation 1.

$$T_{serial} = \sum_{i=0}^{n-1} \tau_i + \sum_{i=0}^{n-1} (D^H_{i(in)} + D^H_{i(out)})\theta \quad (1)$$

where $D^H_{i(in)}$ and $D^H_{i(out)}$ are the total amount of the kernels' input data produced by the host and of the kernels' output data consumed by the host.

When instruction parallelism is exploited with depth $m$ (assume that $m \geq n$), the total execution time is approximated by Equation 2.

$$\begin{aligned} T_{pipeline} =& (\frac{\tau_0}{m} + O_0) + \max_{0 \leq i \leq 1} (\frac{\tau_i}{m} + O_i) + ... \\ &+ \max_{0 \leq i \leq n-1} (\frac{\tau_i}{m} + O_i) \times (m - n + 1) \\ &+ \max_{1 \leq i \leq n-1} (\frac{\tau_i}{m} + O_i) + ... + \max_{n-2 \leq i \leq n-1} (\frac{\tau_i}{m} + O_i) \\ &+ (\frac{\tau_{n-1}}{m} + O_{n-1}) + \sum_{i=0}^{n-1} (D^H_{i(in)} + D^H_{i(out)})\theta \end{aligned} \quad (2)$$

where $O_i$ is the overhead explained in the previous section. The instruction parallelism is beneficial when $T_{pipeline} < T_{serial}$.

Figure 7 shows a comparison between serial processing and instruction parallelism processing. In the instruction parallelism processing approach, the number of pipeline stage, the number of kernels, is 3 ($n = 3$) while the pipeline depth, the number of data segments, is 4 ($m = 4$).

### C. Design algorithm

This section presents the proposed algorithm using quantitative data communication to implement a specific application on an existing HW accelerator platform. The algorithm results in a HW accelerator system with an optimized hybrid interconnect in terms of communication time while keeping HW resources

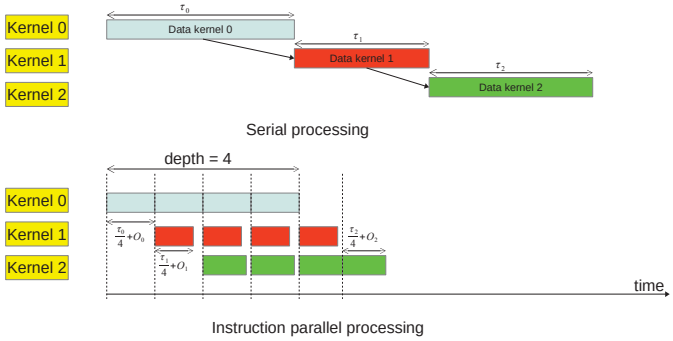


Fig. 7. An example of instruction parallelism processing compared to serial processing

usage for the interconnect minimal. The hybrid interconnect includes the above mentioned interconnect components.

Algorithm 1 shows the pseudo code of the proposed algorithm. The algorithm, first, selects functions which are the most computationally intensive and suitable for accelerating on HW fabrics (i.e., those functions that can be implemented in HW) (line 1). The most computationally intensive functions are considered for data parallelism if acceptable (line 2-8). The algorithm, then, uses the QUAD profiling tool to generate the quantitative data communication profiling of the application (line 9). Based on this profiling, an efficient hybrid interconnect is defined.

**Algorithm 1** System design

**Input:** Application source code
**Output:** A HW accelerator system with an optimized interconnect

1: $L_{hw} \leftarrow$ List of the most computationally intensive functions suitable to implement on HW
2: **repeat**
3: **for** each $HW$ in $L_{hw}$ **do**
4: **if** $HW$ satisfies the data parallelism ($\Delta_{dp} > 0$) & resource is available **then**
5: Replicate $HW$ in $L_{hw}$
6: **end if**
7: **end for**
8: **until** $L_{hw}$ does not change
9: $G \leftarrow$ Quantitative data communication profiling for functions in $L_{hw}$
10: **for** each data communication in $G$ **do**
11: **if** Shared local memory conditions are asserted **then**
12: Apply the shared local memory solution for the two kernels of this data communication
13: Remove the sending kernel from $L_{hw}$
14: **end if**
15: **end for**
16: Map all $HW$ in $L_{hw}$ to the NoC as proposed in Section IV-A2
17: Apply instruction parallelism for all kernels if possible

In this algorithm, shared local memory (line 10-15) is investigated first as explained in Section IV-A1. This communication can also be performed 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 6× larger than the HW resources usage for shared local memory (in the Xilinx xc5vfx130t FPGA device, the four routers require 1221 Look-up Table (LUT) while it is 201 LUT for the crossbar). Shared local memory represents an optimized solution compared to the NoC in terms of HW resources usage. Therefore, it is considered before the NoC. The next step is to map all the remaining kernels which are not connected using the shared local memory solution to the NoC (line 16) as presented in Section IV-A2. Finally, instruction parallelism is considered to further reduce execution time if acceptable (line 17).

## V. Experiments

In this section, we present our experimental results in both an embedded system and a high-performance computing system. We analyze system performance, HW resources usage, and energy consumption.

### A. Experimental setup

The Molen system implemented in the Xilinx ML510 board [47] is used as the embedded platform while the Convey HC2-ex [8] machine is used as our experimental high-performance computing platform. In order to implement our proposed hybrid interconnect for the kernels, we develop a 2×2 crossbar for shared local memory and adapt the NoC presented in [48] into our systems.

In the Molen platform, the PowerPC processor - an embedded hardwired processor of the FPGA device - acts as the host processor while accelerator kernels are mapped onto the reconfigurable area of the device. SDRAM memory connected directly to the PowerPC through a Xilinx core is the main memory of the system. While the host processor works at 400MHz, the kernels work at 100MHz. The Xilinx PLB bus is used as the communication infrastructure which connects the host processor, the kernels and other modules such as I/O, Interrupt, Timer, etc., together. The Convey HC-2ex system consists of one host Intel Xeon X5670 processor and four Virtex-6 xc6vlx760 FPGA devices where kernels are mapped on. While the host that consists of 6 physical hyper-threading cores works at 2.93GHz, the kernels works at 150MHz. The host processor and the accelerator kernels can communicate through a Hybrid-core Globally Shared Memory (HGSM) - the communication infrastructure - controlled by a Convey's HCMI. In both systems, BRAM inside the FPGA devices are used as local memory of accelerator kernels.

Our experiments use two applications: Canny edge detection [49] and KLT feature tracker [50]. We first run these applications on host processors of both systems to get SW execution time. In the Molen system, the single core PowerPC performs these applications. In the Convey system, the functions that are accelerated on the FPGA devices are processed by all the 12 cores of the host processor using the OpenMP library, i.e. the host processor is fully utilized. The applications are compiled by GCC 4.2 with $-O2$ optimization level.

Thereafter, we develop these applications on both systems without our hybrid interconnect, called baseline systems. In these systems, data communication is carried out by the system

communication infrastructure, the PLB bus in the Molen platform and HGSM in the Convey machine. The DWARV [44] compiler automatically generates the HDL description for the kernels on Molen from their C code while Xilinx Vivido HLS is used for kernels on Convey. We use different high level synthesis tools to illustrate that our proposed approach is flexible and compatible with different design tools. The systems are synthesized with Xilinx ISE without any manual optimization.

Finally, we implement the applications on both systems using our proposed approach. The next section will compare our systems to baseline systems and SW in terms of system performance, HW resources usage and energy consumption parameters.

### B. Experimental results

In this section, we analyze system performance, HW resources usage and energy consumption for both systems with the two applications.

*1) System performance analysis:* Table II shows the speed-up of our proposed systems compared to both SW and the baseline systems in the embedded and the high-performance computing platforms. As shown in the table, when the proposed approach and the hybrid interconnect are exploited, it achieves speed-ups of the overall application by up to 3.72× in the embedded platform and by up to 1.55× in the high-performance computing platforms compared to SW. Compared to the baseline system, overall application speed-ups of up to 1.83× for the embedded system and 1.53× for the high-performance computing platform are obtained.

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

| Plat. | App. | #Kernels (#Func.) | w.r.t Software | | w.r.t Baseline | |
|---|---|---|---|---|---|---|
| | | | App. | Kernels | App. | Kernels |
| **Molen** | Canny | 5 (4) | 3.15× | 3.88× | 1.83× | 2.12× |
| | KLT | 3 (3) | 3.72× | 6.58× | 1.26× | 1.55× |
| **Convey** | Canny | 64 (4) | 1.55× | 2.20× | 1.53× | 2.17× |
| | KLT | 56 (3) | 1.02× | 1.13× | 1.20× | 2.50× |

**Legend**: Plat.: Platform; App.: Application; Func.: the number of functions that are accelerated by HW accelerator kernels

*2) Hardware resources usage:* Table III presents the HW resources utilization of the baseline, our proposed systems and the NoC-only systems, in terms of the number of FPGA look-up tables (LUTs) and the number of FPGA registers. NoC-only system is a system in which only NoC is used for the interconnect of kernels. All kernels and local memories are connected to the NoC instead of using communication pattern-based mapping as proposed in Section IV-A2. As shown in the table, our systems 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 the baseline systems or NoC-only system. The baseline systems are low performance while the NoC-only systems use more HW resources than ours. Meanwhile, our systems achieve the same performance and uses less resources than the NoC-only systems. Figure 8 presents the comparison of HW resources used for interconnect and for the kernels in our system normalized to the resources used for computing (kernels). The interconnect uses only 49.2% resources compared to the resources used for computing at most.

TABLE III
HW RESOURCES UTILIZATION (#LUTS/#REGISTERS)

| Plat. | App. | Baseline | Pro. | NoC. | Red. | Solution |
|---|---|---|---|---|---|---|
| Molen | Canny | 9926<br>12707 | 15227<br>18657 | 17894<br>21059 | 14.9%<br>11.4% | NoC,<br>SM, P |
| | KLT | 4721<br>5430 | 4921<br>5631 | 7358<br>8070 | 33.1%<br>30.2% | SM |
| Convey | Canny | 74965<br>48994 | 90789<br>54849 | 93693<br>58421 | 3.1%<br>6.1% | NoC,<br>SM, P |
| | KLT | 106162<br>95804 | 107919<br>96664 | 118083<br>109116 | 8.6%<br>11.4% | SM, P |

**Legend**: Plat.: Platform; App.: Application; Pro.: Proposed; NoC.: NoC-only system; Red.:Reduction to NoC-only system; SM: Shared memory; P: Parallelism

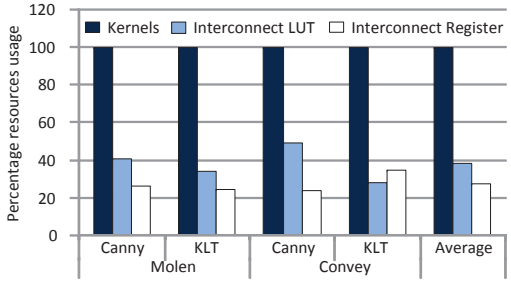


Fig. 8. Interconnect resources usage normalized to kernels resources usage

*3) Energy consumption:* To compute energy consumption, we use Xilinx Power Analyzer to estimate the power consumption of these systems. Figure 9(a) compares energy consumption of overall applications on the Molen platform between the baseline system and our proposed system normalized to baseline energy consumption. According to the figure, our proposed system saves up to 50.3% energy consumption compared to baseline system.

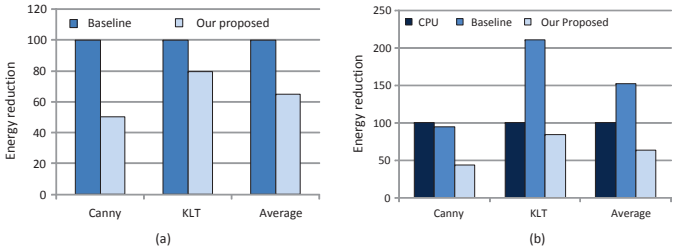


Fig. 9. (a) Overall application energy consumption reduction of our proposed system compared to baseline system in the Molen platforms; (b) Kernels energy consumption comparison in the Convey platforms normalized to the host processor energy consumption.

With the high-performance computing platform, we compare energy consumption of accelerator kernels instead of the whole system. Since the Convey machine has four FPGAs, energy consumption for kernels in each application is approximated by four times the product of power consumption and execution time. We compare our proposed system to both baseline system and host processor. Figure 9(b) shows the reduction in energy of our system normalized to the host processor energy consumption. As shown in the figure, the

KLT application implemented in the baseline system uses more energy than the host processor. Our proposed system consumes less energy than both the host processor and the baseline system. Up to 54.2% energy consumption is saved when compared to baseline system.

## VI. Conclusion

In this paper, we presented an automated design approach to implement a specific application in an existing hardware accelerator system with an efficient hybrid interconnect for kernels using quantitative data communication profiling of the application. The hybrid interconnect includes a NoC, shared local memory solution, or both. To further optimize system performance, parallelizing kernel processing was taken into consideration. We developed our experiments on both the Molen embedded platform and the Convey high-performance computing platform. We compared our proposed systems with the original systems as well as the software running on the PowerPC at 400MHz in the Molen platform and on the 12 cores Intel Xeon processor in the Convey platform. The results showed that in both platforms, we achieved overall application speed-ups compared to the baseline systems. Moreover, due to the reduction in execution time, our systems also used less energy compared to the baseline systems.

## References


[1] S. Leibson, "Whats the Right Road to ASIC-Class Status for FPGAs?" May 2014.

[2] R. Kumar, et al., "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, Nov. 2005.

[3] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, no. 5, pp. 67–77, May 2011.

[4] H. Esmaeilzadeh, et al., "Dark silicon and the end of multicore scaling," *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 365–376, Jun. 2011.

[5] S. Vassiliadis, et al., "The molen polymorphic processor," *Computers*, vol. 53, no. 11, pp. 1363–1375, Nov 2004.

[6] N. S. Voros, et al., "Morpheus: A heterogeneous dynamically reconfigurable platform for designing highly complex embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 3, pp. 70:1–70:33, Apr. 2013.

[7] O. Pell and O. Mencer, "Surviving the end of frequency scaling with reconfigurable dataflow computing," *SIGARCH Comput. Archit. News*, vol. 39, no. 4, pp. 60–65, Dec. 2011.

[8] Convey Computer, "Convey reference manual," 2012.

[9] J. Stuecheli, "Next generation power microprocessor," in *HotChips 2013*, August 2013.

[10] A. Putnam, et al., "A reconfigurable fabric for accelerating large-scale datacenter services," in *ISCA*, June 2014.

[11] M. Rutzig, et al., "Multicore platforms: Processors, communication and memories," in *Adaptable Embedded Systems*. Springer, 2013, pp. 243–277.

[12] R. Lysecky and F. Vahid, "Design and implementation of a microblaze-based warp processor," *ACM Trans. Embed. Comput. Syst.*, vol. 8, no. 3, pp. 22:1–22:22, Apr. 2009.

[13] T. Schumacher, et al., "IMORC: An infrastructure and architecture template for implementing high-performance reconfigurable FPGA accelerators," *Microprocessors and Microsystems*, vol. 36, no. 2, pp. 110 – 126, 2012.

[14] A. Ismail and L. Shannon, "Fuse: Front-end user framework for o/s abstraction of hardware accelerators," in *FCCM*, May 2011, pp. 170–177.

[15] C. Pilato, et al., "On the automatic integration of hardware accelerators into fpga-based embedded systems," in *FPL*, Aug 2012, pp. 607–610.

[16] A. Canis, et al., "Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2, pp. 24:1–24:27, Sep. 2013.

[17] T. Heil, et al., "Architecture and performance of the hardware accelerators in ibm's poweren processor," *ACM Trans. Parallel Comput.*, vol. 1, no. 1, pp. 5:1–5:26, May 2014.

[18] A. Grasset, et al., "The morpheus heterogeneous dynamically reconfigurable platform," *International Journal of Parallel Programming*, vol. 39, no. 3, pp. 328–356, 2011.

[19] E. S. Chung, et al., "Coram: An in-fabric memory architecture for fpga-based computing," in *FPGA*. New York, NY, USA: ACM, 2011, pp. 97–106.

[20] ——, "Prototype and evaluation of the coram memory architecture for fpga-based computing," in *FPGA*. New York, NY, USA: ACM, 2012, pp. 139–142.

[21] L. Benini, et al., "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *DATE*, March 2012, pp. 983–987.

[22] L. Ling, et al., "High-performance, energy-efficient platforms using in-socket fpga accelerators," in *FPGA*, 2009, pp. 261–264.

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

[24] B. Betkaoui, et al., "A framework for fpga acceleration of large graph problems: Graphlet counting case study," in *FPT*, 2011, pp. 1–8.

[25] J. Cong and B. Xiao, "Optimization of interconnects between accelerators and shared memories in dark silicon," in *Computer-Aided Design*, Nov 2013, pp. 630–637.

[26] D. Sanchez, et al., "An analysis of on-chip interconnection networks for large-scale chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 1, pp. 4:1–4:28, May 2010.

[27] J. Hur, "Customizing and hardwiring on-chip interconnects in fpgas," Ph.D. dissertation, Delft University of Technology, Delft, Netherlands, February 2011.

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

[29] T. Richardson, et al., "A hybrid soc interconnect with dynamic tdma-based transaction-less buses and on-chip networks," in *VLSI Design*, Jan 2006, pp. 8 pp.–.

[30] B. Grot, et al., "Express cube topologies for on-chip interconnects," in *HPCA*, Feb 2009, pp. 163–174.

[31] R. Manevich, et al., "Best of both worlds: A bus enhanced noc (benoc)," in *Networks-on-Chip*, May 2009, pp. 173–182.

[32] R. Das, et al., "Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps," in *HPCA 2009*, Feb 2009, pp. 175–186.

[33] A. Avakian, et al., "A reconfigurable architecture for multicore systems," in *IPDPSW*, April 2010, pp. 1–8.

[34] K.-L. Tsai, et al., "Design of low latency on-chip communication based on hybrid noc architecture," in *NEWCAS*, June 2010, pp. 257–260.

[35] P. Zarkesh-Ha, et al., "Hybrid network on chip (hnoc): Local buses with a global mesh architecture," in *System Level Interconnect Prediction*. New York, NY, USA: ACM, 2010, pp. 9–14.

[36] H. Giefers and M. Platzner, "A triple hybrid interconnect for many-cores: Reconfigurable mesh, noc and barrier," in *FPL*, Aug 2010, pp. 223–228.

[37] Y. Jin, et al., "Communication-aware globally-coordinated on-chip networks," *Parallel and Distributed Systems*, vol. 23, no. 2, pp. 242–254, Feb 2012.

[38] H. Zhao, et al., "A hybrid noc design for cache coherence optimization for chip multiprocessors," in *DAC*. New York, NY, USA: ACM, 2012, pp. 834–842.

[39] V. Todorov, et al., D. Mueller-Gritschneder, H. Reinig, and U. Schlichtmann, "Deterministic synthesis of hybrid application-specific network-on-chip topologies," *Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 10, pp. 1503–1516, Oct 2014.

[40] J. Cong and Y. Zou, "Fpga-based hardware acceleration of lithographic aerial image simulation," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, no. 3, pp. 17:1–17:29, Sep. 2009.

[41] S. Ostadzadeh, "Quantitative application data flow characterization for heterogeneous multicore architectures," Ph.D. dissertation, Delft University of Technology, Delft, Netherlands, December 2012.

[42] G. S. L., "Gprof: A call graph execution profiler," *SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, Jun. 1982.

[43] Xilinx, "Vivado high-level synthesis," 2014.

[44] R. Nane, et al., "Dwarv 2.0: A cosy-based c-to-vhdl hardware compiler," in *FPL*, Aug 2012, pp. 619–622.

[45] J. Choi, et al., 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*, April 2012, pp. 17–24.

[46] J. L. Gustafson, "Reevaluating amdahl's law," *Commun. ACM*, vol. 31, no. 5, pp. 532–533, May 1988.

[47] Xilinx, "Ml510 reference design," 2009.

[48] J. Heisswolf, et al., "A scalable noc router design providing qos support using weighted round robin scheduling," in *ISPA*, July 2012, pp. 625–632.

[49] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov 1986.

[50] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition*, Jun 1994, pp. 593–600.