Rule-Based Data Communication Optimization Using Quantitative 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*—Multicore architectures, especially hardware accelerator systems with heterogeneous processing elements, are being increasingly used due to the increasing processing demand of modern digital systems. However, data communication in multicore architectures is one of the main performance bottlenecks. Therefore, reducing data communication overhead is an important method to improve the speed-up of such systems. In this paper, we propose a heuristic-based approach to address the data communication bottleneck. The proposed approach uses a detailed quantitative data communication profiling to generate interconnect designs automatically that are relatively simple, low overhead and low area solutions. Experimental results show that we can gain speed-up of $3.05\times$ for the whole application and up to $7.8\times$ speed-up for accelerator functions in comparison with software.**

***Index Terms*—hardware accelerator, data communication bottleneck, quantitative profiling, design rules.**

I. Introduction

Multicore architectures are being increasingly used in modern digital systems. Multicore architectures can be seen as Multiprocessor System-on-Chip (MPSoC) in which more than one processing element (PEs) (or called computational cores) and memory systems are integrated on a single chip. One design approach for a multicore system is hardware software co-design (called hardware accelerator system) due to the straightforward and easy debugging of software and potential energy and performance benefits of hardware. In such system, some application segments execute on software (general purpose processor) and other segments are synthesized into custom circuits, so-called hardware accelerators (usually FPGAs).

In these systems, data is usually needed to transfer between the main memory and the local memories of hardware accelerators. In image processing and multimedia systems, which process large amounts of data, this data communication takes a large portion of application execution time. Therefore, data communication usually is a primary anticipated bottleneck for system performance [1], [2], [3], [4]. Consequently, one important method to improve the speed-up of such systems is reducing data communication overhead.

Reducing data communication overhead can be done by increasing communication throughput or decreasing the amount of data movement from one memory to another memory. Much research on bus-based architectures as well as Network-on-Chip-based architectures to improve the throughput of data communication has been done in recent years such as in [1], [2], [3], [5], [6]. The second approach is by decreasing the amount of data movement from one memory to another by delivering data in-place. The work in [4] proposed the Remote DMA technique to deliver application data to the exact memory location.

However, aforementioned proposals use static information of applications (such as task graphs) to approach the interconnect. In our work, the quantitative data communication profiling is used to approach the optimized interconnect solutions for the specific application under consideration. The ultimate goal is to have an automatic design taking runtime communication pattern into account such that the most appropriate interconnect infrastructure is available. This paper is the first step using a rule-based approach to have an appropriate application specific interconnect infrastructure. In this work, we use two techniques to reduce data communication overhead: 1) Using *shared hardware accelerator local memory* to eliminate the data movement between the local memories; and 2) Using *pipelining data communication* to parallelize the execution of hardware accelerator functions and the data communication to hide the data communication overhead.

The main contributions of this paper can be summarized as follows: 1) the utilization of quantitative data communication profiling to address the data communication bottleneck in hardware accelerator systems; 2) the introduction of a rule-based and detailed profiling driven interconnect design with an emphasis on runtime management; 3) the presentation of experimental results on five different applications on a real FPGA platform.

The rest of the paper is organized as follows. Section II briefly describes the research context of the work presented in this paper. Section III presents in details our approaches to reducing data communication overhead. Section IV shows our design strategies for a specific application using profiling information. Subsequently, a case study of image processing application is illustrated in Section V. We discuss our experimental results in Section VI. Finally, Section VII concludes the paper and presents the future work.

II. Research Context

A. The Molen architecture

The Molen architecture [7] (depicted in Figure 1), is a heterogeneous, shared memory multicore system for software/hardware co-design. The Molen architecture consists of two types of PE: one *General Purpose Processor* (GPP) and one or more *Reconfigurable Processor(s)*, called Custom Computing Unit(s) (CCUs). GPP uses an Auxiliary Processor Unit (APU) to control the execution of CCUs. Each CCU has each local memory (CCUMem) to store the data. CCU exchanges parameters with GPP via exchange registers (CCUXreg). CCUs are usually implemented on a reconfigurable area such as an FPGA for accelerating some functions which cannot be implemented or compatible with the GPP. The main application is executed on GPP. The main advantage

978-1-4673-2845-6/12/$31.00 © 2012 IEEE

of the Molen architecture is that it can be ported to various reconfigurable platforms easily. We use the Molen architecture, whose current version on Xilinx virtex 5 FPGA can support a maximum of five accelerator functions (due to the available reconfigurable area), to conduct our experiment.

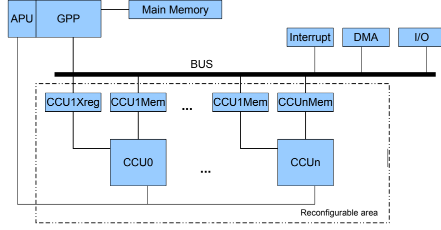


Fig. 1. The Molen architecture

B. *Profiling tools*

The QUAD toolset [8] 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 a Quantitative Data Usage (QDU) 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.

III. Communication Acceleration Solutions

A. *Definitions and assumptions*

Before presenting possible solutions for communication acceleration, we need to define some equations used to estimate the quality of the solutions. The following vocabulary is used:

- **Hardware accelerator function** is defined by *Function*(H, D_i, D_o); where H is the computation time of the hardware accelerator only, D_i and D_o is the total amount of data input and output in bytes.
- **Data communication** between two functions is defined by $C_{ij}(F_i, F_j, D_{ij})$; where F_i and F_j are the producer and the consumer function, respectively, and D_{ij} is the total amount of data in bytes transferred from F_i to F_j.
- **The average time** taken by the GPP or the DMA for transferring 1 byte between the main memory and a hardware accelerator local memory is t_g or t_d, respectively. These values are platform dependent, however $t_d < t_g$.

Hardware accelerator systems, such as Molen, LegUp [9], usually use the heterogeneous memory system where the GPP has the main memory and the hardware accelerators have their local memories. In our discussion, we assume that while the GPP can access the main memory as well as the local memories of hardware accelerators, hardware accelerators can access their own memories only.

B. *Shared hardware accelerator local memory*

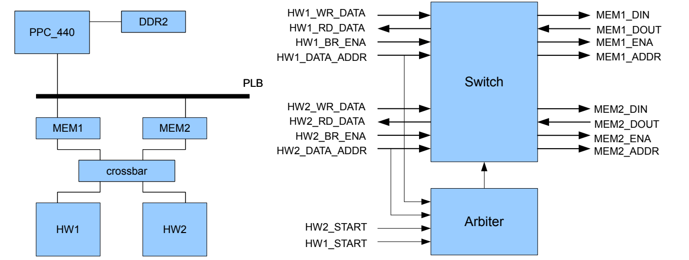
Consider two hardware accelerators $HW_1(H_1, D_{1i}, D_{1o})$ and $HW_2(H_2, D_{2i}, D_{2o})$ and the data communication $C_{12}(HW_1, HW_2, D_{12})$. In typical data communication, the GPP copies data from the main memory to the local memory of HW_1 first. It then copies the result of HW_1 to the main memory when HW_1 is finished. Because HW_2 requires an amount D_{12} of data from HW_1, the GPP has to copy D_{12} bytes of the result of HW_1 and other required data from the main memory to the local memory of HW_2. The total execution time for HW_1 and HW_2 is as in Equation 1.

$$T = H_1 + H_2 + (D_{1i} + D_{1o} + D_{2i} + D_{2o})t_g \qquad (1)$$

Assume that HW_2 requires data from two sources: (1) D_{12} bytes from HW_1, and (2) $D_n = D_{2i} - D_{12}$ bytes from the main memory (produced by other functions). The GPP can copy D_n bytes of data input from the main memory to the local memory of HW_2 in parallel with the execution of HW_1. Then DMA is used to transfer the D_{12} bytes of data from the local memory of HW_1 to the local memory of HW_2. The part of the result of HW_1 used by other functions rather than HW_2 is copied back to the main memory in parallel with the execution of HW_2. The total execution time of the two hardware accelerators is as in Equation 2.

$$T_{dma} = H_1 + H_2 + (D_{1i} + D_{2o})t_g + D_{12}t_d \qquad (2)$$

Using DMA not only has a hardware overhead but also does not hide all data communication time for the two hardware accelerators. In this work, we propose to use a crossbar between hardware accelerators communicating together and their local memories. Figure 2a illustrates a simple system with the two hardware accelerators HW_1 and HW_2 sharing their local memories using a crossbar based on the Molen architecture. Figure 2b depicts the detailed structure of the crossbar for the Molen hardware accelerator functions.



(a) HW_1 and HW_2 share their memories using a crossbar

(b) Structure of the crossbar

Fig. 2. Shared hardware accelerator local memory

With the crossbar between the two hardware accelerators, HW_1 can access not only its own local memory but also the local memory of HW_2 and vice versa. Therefore, neither GPP nor DMA is needed for data communication between them. The total execution time for the two hardware accelerators is as in Equation 3.

$$T_{crossbar} = H_1 + H_2 + (D_{1i} + D_{2o})t_g \qquad (3)$$

Using the crossbar, we can reduce the time $\Delta = T - T_{crossbar} = (D_{1o} + D_{2i})t_g$ in comparison with the typical data communication model. In addition, the hardware overhead of our crossbar is less than the DMA in our target hardware system (see Table I). Using the data communication profiling from QUAD, we can decide which hardware accelerators should share their local memories.

C. *Pipelining data communication*

The previous section shows how a crossbar reduces the communication time between the two hardware accelerators.

This section presents a pipeline data communication solution to hide the data communication overhead. In some specific applications, especially in multimedia application such as image or video processing, data can be processed as streaming input. Using this concept, we can reduce data communication time by segmenting the input data and running the hardware accelerator on each data segment independently.

Consider again two hardware accelerator functions HW_1 and HW_2. Assume that these functions can process segments of input data that can be used to synthesized the final result. Because the two hardware accelerators can process segments of input data, HW_1 can start as soon as the first input data segment is copied.

Assume that we divide the input data for HW_1 into two different segments S_1 and S_2. The processing of GPP and the two hardware accelerators follows the pseudo code in Algorithm 1.

Algorithm 1 Pipelining data communication

1: GPP copies S_1 from the main memory to HW_1 local memory;
2: HW_1 processes S_1 while GPP copies S_2 from the main memory to HW_1 local memory in parallel;
3: DMA transfers result of S_1 from HW_1 to HW_2 local memory;
4: HW_1 processes S_2 while HW_2 processes the first segment in parallel;
5: DMA transfers result of S_2 from HW_1 to HW_2 local memory;
6: HW_2 processes the second segment while GPP copies final result of the first segment from HW_2 local memory to the main memory in parallel;
7: GPP copies final result of the second segment from HW_2 local memory to the main memory;

The total execution time of the two hardware accelerators in this case is as in Equation 4.

$$T_p = \frac{H_1}{2} + max(\frac{H_1}{2}, \frac{H_2}{2}) + \frac{H_2}{2} + (\frac{D_{1i}}{2} + \frac{D_{2o}}{2})t_g + D_{12}t_d + O \quad (4)$$

where O is the overhead for processing streaming input. If $O + D_{12}t_d < min(\frac{H_1}{2}, \frac{H_2}{2}) + (\frac{D_{1i}}{2} + \frac{D_{2o}}{2})t_g$, we have $T_p < T_{crossbar}$. The pipelining data communication should be applied for the two hardware accelerators rather than shared local memory.

IV. Data Communication Profiling Driven Design Strategies

In this section, we introduce strategies to design an application-specific system using the above mentioned hardware techniques. To decide which techniques should be applied in a system for a specific application, we propose four design rules (heuristics). These rules identify the most optimal hardware data communication solution based on the data communication profiling provided by the QUAD tool.

In the previous section, we introduce two different hardware techniques for increasing the speed-up of hardware accelerator functions, those are shared local memories of hardware accelerators using crossbar and pipelining the data communication. In addition, hardware duplication of the most time-consuming function can also be used to improve performance of the system.

In order to choose which techniques should be used for a specific application, we propose four decision rules to identify the most optimal solution. Based on these rules, Algorithm 2 builds a hardware accelerator system for the specific application with the most optimized interconnect.

Algorithm 2 Data communication profiling-driven design

Input: Hardware accelerators execution time, Data communication profiling
Output: A hardware accelerator system with the most optimized interconnect

1: Check Rule 1 for the most computationally-intensive HW accelerator;
2: **for** each data communication between accelerators **do**
3: Check Rule 2, Rule 3 and Rule 4;
4: **end for**
5: **return** A hardware accelerator system with the most optimized interconnect

Rule 1. *Hardware duplication*

Assume that the most computationally-intensive hardware accelerator $F_1(H_1, D_{1i}, D_{1o})$ can be executed in parallel with different data input segments with the overhead O. The execution time of $F1$ in the non-optimized case is as follows

$$T = H_1 + (D_{1i} + D_{1o})t_g \quad (5)$$

Assume that H_2 is the execution time of the second most computationally-intensive hardware accelerator. *The hardware accelerator of the most computationally-intensive function is duplicated twice if $H_1 >= 2H_2$ and $O < \frac{H_1}{2}$; and DMA is used to transfer data between the two duplicated hardware accelerators with other hardware accelerators.*

In this case, the total execution time of F_1 is as in Equation 6, where D_{dma} is the total amount of data transfer from the duplicated hardware accelerators to other hardware accelerators using DMA.

$$T_1 = \frac{H_1}{2} + (D_{1i} + D_{1o} - D_{dma})t_g + D_{dma}t_d + O \quad (6)$$

Rule 2. *Pipelining data communication application*

Consider two hardware accelerator functions $F_1(H_1, D_{1i}, D_{1o})$ and $F_2(H_2, D_{2i}, D_{2o})$ communicating together with the communication $C_{12}(F_1, F_2, D_{12})$, assume that they can be executed in parallel with streaming data input with the overhead O:

The pipelining to data communication is applied if $O + D_{12}t_d < min(\frac{H_1}{2}, \frac{H_2}{2}) + (\frac{D_{1i}}{2} + \frac{D_{2o}}{2})t_g$; otherwise the shared local memory is applied.

Rule 3. *Shared local memory application*

Consider two hardware accelerator functions $F_1(H_1, D_{1i}, D_{1o})$ and $F_2(H_2, D_{2i}, D_{2o})$ communicating together with the communication $C_{12}(F_1, F_2, D_{12})$:

The shared local memory is applied if the two hardware accelerators cannot be executed in parallel.

In the case that the two hardware accelerators can be executed in parallel with the overhead O. *The shared local memory is also applied if $O + D_{12}t_d > min(\frac{H_1}{2}, \frac{H_2}{2}) + (\frac{D_{1i}}{2} + \frac{D_{2o}}{2})t_g$.*

Rule 4. *3-function communication*

Consider three hardware accelerator functions $F_1(H_1, D_{1i}, D_{1o})$, $F_2(H_2, D_{2i}, D_{2o})$ and $F_3(H_3, D_{3i}, D_{3o})$ which have the communication topology as depicted in Figure 3. The communication among these functions is as follows: $C_{12}(F_1, F_2, D_{12})$, $C_{13}(F_1, F_3, D_{13})$ and $C_{23}(F_2, F_3, D_{23})$.

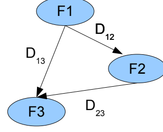


Fig. 3. 3-function communication

The total execution time of these hardware accelerator functions in non-optimized case is:

$$T = H_1 + H_2 + H_3 + (\sum_{j=1}^{3}(D_{ji} + D_{jo}))t_g \quad (7)$$

1) If $D_{12} > D_{23}$, the crossbar is used between F_1 and F_2; and DMA is used to transfer data from F_2 to F_3 right after F_2 is finished as well as from F_1 to F_3 in parallel with the execution of F_2. The total execution time is:

$$T_4^{(1)} = H_1 + H_2 + H_3 + D_{1i}t_g + D_{23}t_d + D_{3o}t_g \quad (8)$$

2) Otherwise, the crossbar is used between F_2 and F_3; and DMA is used to transfer data from F_1 to F_2 as well as from F_1 to F_3 in parallel with the execution of F_2. The total execution time is:

$$T_4^{(2)} = H_1 + H_2 + H_3 + D_{1i}t_g + D_{12}t_d + D_{3o}t_g \quad (9)$$

In this case, we have $T_4^{(1)} < T$ and $T_4^{(2)} < T$. The reduction in time is $\delta = (D_{1o}+D_{2i}+D_{2o}+D_{3i})t_g - min(D_{12}, D_{23})t_d$.

V. Case Study

The previous sections presented the hardware techniques and the design rules using data communication profiling provided by QUAD to obtain the most optimized interconnect solution for each application. In this section, we present a case study to clarify the introduced techniques as well as design rules.

Our case study uses the Canny edge detection application. The Canny application [10] is a well-known edge detection algorithm. In this work, we use the implementation version provided by the University of South Florida [11]. A grayscale *PGM* image with resolution 100×133 pixels with 8 bits per pixel is used in the experiment. The most time-consuming functions in the specific application are targeted for hardware accelerators and inputted to the DWARV tool [12] to generate VHDL description. The next step is to use QUAD tool to generate a QDU (Quantitative Data Usage) graph as presented in details in Section II-B. Figure 4 presents the QDU graph generated profiling tools for the Canny application.

The information from *gprof* tool [13] shows that functions *gaussian_smooth*, *derrivative_x_y*, *magnitude_x_y* and *non_max_supp* take the most percentage of the execution time. Therefore, they are targeted to accelerate on hardware. The name of hardware accelerator functions has added prefix "hw_" for distinguishing from software functions. The most computationally-intensive function *gaussian_smooth* takes around $5\times$ longer than the second computationally-intensive function. The overhead for hardware duplication

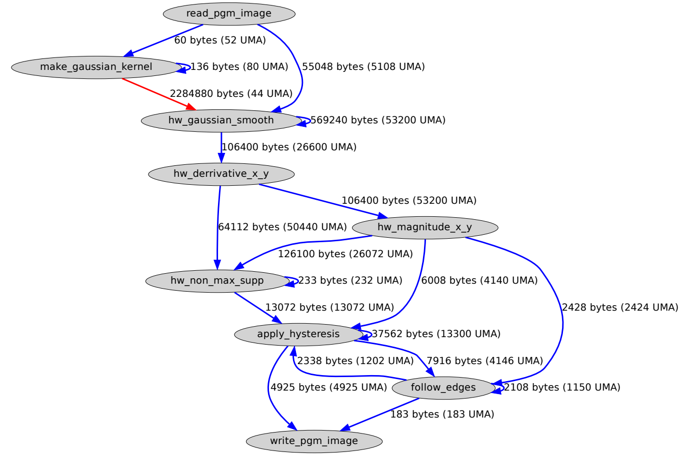


Fig. 4. Profiling graphs for the Canny edge detection application

for this function also satisfies the requirement in Rule 1. Hence, Rule 1 is applied to this hardware accelerator. The communication among *hw_derrivativ_x_y*, *hw_magnitude_x_y* and *non_max_supp* functions is investigated next. Two functions *derrivativ_x_y* and *magnitude_x_y* satisfies the requirement in Rule 2. Hence, Rule 2 is true. The pipelining data communication technique is used to parallelize the execution of *hw_derivative_x_y* and *hw_magnitude_x_y*. Consequently, Rule 3 is applied for the data communication between *hw_magnitude_x_y* and *hw_non_max_supp* hardware accelerators. The crossbar is used to share the local memories of the two hardware accelerators.

The final version of the hardware accelerator system based on the Molen architecture using presented hardware techniques and proposed design rules for Canny application is shown in Figure 5.

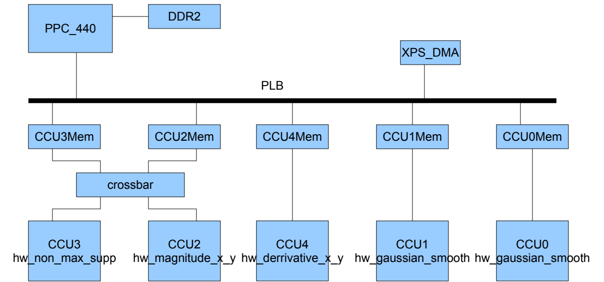


Fig. 5. Final system for our case study based on Molen architecture and proposed design rules

TABLE I
Resource Usage and Maximum Frequency of Hardware Modules

HW Module	Resource (# of LUTs)	Max. frequency
Crossbar	201	N/A
DMA	556	252.717MHz
hw_derrivative_x_y	1463	317.269MHz
hw_magnitude_x_y	971	388.342MHz
hw_gaussian_smooth	1938	264.460MHz
hw_non_max_supp	4959	313.908MHz

Table I summarizes the hardware size and maximum frequency of our CCUs as well as our crossbar. The target hardware system in our case study is Xilinx FPGA ML510 board [14] containing a Xilinx Virtex 5 FPGA xc5vf130t. The PowerPC 440, used as the GPP, and the CCUs are set up to

TABLE II
EXECUTION TIMES OF ACCELERATED FUNCTIONS AND SPEED-UP

Scenario	Execution time	Resource	Speed-up
Software	16,723,007 (41.81ms)	N/A	N/A
Standard Molen	9,033,618 (22.58ms)	9331 LUTs	1.85×
Molen with rules	4,405,894 (11.02ms)	12026 LUTs	3.79×

400MHz and 100MHz, respectively. Table II summarizes the total software times, the total hardware times and the speed-up of the accelerated functions. Row 1 shows the total software times of the functions accelerated on hardware; Row 2 and Row 3 shows the hardware times of the accelerated functions without and with applying our design rules, respectively.

The maximum speed-up for hardware accelerator functions is up to $3.79\times$ when the proposed design rule is applied. The overall application speed-up we can gain is $3.05\times$.

VI. EXPERIMENTAL RESULTS

Following the approach presented in the Canny case study, we implemented four more applications. The first two are from the same image processing domain as Canny and are the Susan edge detector, with an implementation version of Oxford University [15], and KLT feature tracker [16]. The two other applications are Fluid simulation [17] and Blowfish application (a symmetric block cipher) from the CHStone benchmark [18]. Table III present the speed-up and the resource usage for the hardware accelerators and interconnect. Column 2 in the table shows the hardware techniques used for each application based on the design rules.

TABLE III
SPEED-UP AND RESOURCE USAGE FOR HARDWARE ACCELERATORS AND INTERCONNECT

Application (# accelerator)	HW techniques	Resource (# LUTs)	Speed-up
Susan (3)	DMA, Crossbar	21504	2.55×
KLT (3)	Crossbar, Duplication	6553	7.8×
Fluid (2)	Crossbar	12569	1.95×
Blowfish (2)	Crossbar	16444	3.02×

The numbers in the table are generated as follows. We first run the applications on the PowerPC to obtain a reference execution time. Then we run the applications with the hardware techniques described earlier in this paper. As shown in Table III, the speed-up of the hardware accelerators is up to $7.8\times$. Figure 6 shows the comparison of the speed-up of the Molen system with and without using the design rules to choose the most optimal interconnect. As shown in this figure, hardware accelerators which apply the data communication profiling driven communication acceleration solutions provide up to $2\times$ execution time improvement in comparison with the accelerators that do not apply the data communication driven communication acceleration solutions.

VII. CONCLUSION & FUTURE WORK

In this paper, we used quantitative data communication profiling information to build up an optimized data communication infrastructure for specific application. We proposed profiling driven design rules based on two data communication acceleration techniques: 1) shared hardware accelerator local memory, and 2) pipelining data communication. These rules allow for runtime hardware accelerator interconnect management which enables selecting the best communication infrastructure available rather than having a fixed one.

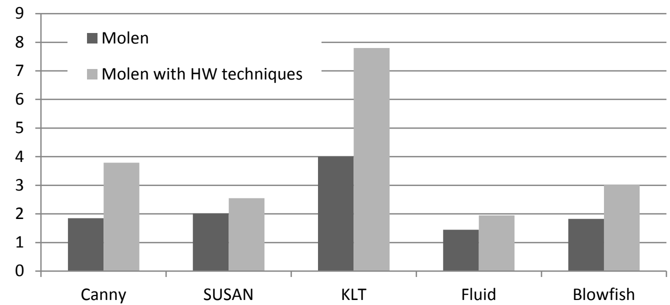


Fig. 6. Speed-up of Molen with and without using design rules

Experimental results show that the speed-up of hardware accelerated functions can go up to $7.8\times$ in comparison with their execution time on PowerPC. The overall application speedup is up to $3.05\times$. Our experimental results show that our rule-based approach can gain speed-up by a factor of 2 in comparison with typical data communication approaches.

In the future, we plan to include different memory hierarchies for validating the approach developed in this paper. In addition, other data communication mechanisms, such as NoC, are also considered.

ACKNOWLEDGMENT

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

REFERENCES

[1] B. Donchev, G. Kuzmanov, and G. N. Gaydadjiev, "External memory controller for Virtex II Pro," in *System-on-Chip*, 2006, pp. 1–4.
[2] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and partial FPGA exploitation," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 438–452, 2007.
[3] C. Altera, "Accelerating Nios II systems with the C2H compiler tutorial," August 2008.
[4] 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.
[5] M. B. Stensgaard and J. Sparso, "ReNoC: A network-on-chip architecture with reconfigurable topology," in *Networks-on-Chip*, 2008, pp. 55–64.
[6] C. Jackson and S. J. Hollis, "Skip-links: A dynamically reconfiguring topology for energy-efficient NoCs," in *System on Chip*, 2010, pp. 49–54.
[7] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. M. Panainte, "The MOLEN polymorphic processor," *IEEE Transactions on Computers*, vol. 53, no. 11, pp. 1363–1375, 2004.
[8] S. A. Ostadzadeh, M. Corina, C. Galuzzi, and K. Bertels, "tQUAD - memory bandwidth usage analysis," in *International Conference on Parallel Processing*, 2010, pp. 217–226.
[9] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "LegUp: high-level synthesis for FPGA-based processor/accelerator systems," in *FPGA*, 2011, pp. 33–36.
[10] J. Canny, "A computational approach to edge detection," *Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679 –698, 1986.
[11] U. S. Florida, "Canny edge detector."
[12] 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.
[13] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," *SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, 1982.
[14] Xilinx, "Ml510 reference design," June 23 2009.
[15] S. M. Smith, "Susan low level image processing," 1992.
[16] J. Shi and C. Tomasi, "Good Features to Track," in *CVPR*, 1994.
[17] J. Stam, "Real-time fluid dynamics for games," in *Game Developer Conference*, 2003.
[18] S. H. Y. Hara, H. Tomiyama and H. Takada, "Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-Based high-level synthesis," *JIP*, vol. 17, pp. 242–254, Oct. 2009.