IEICE TRANS. FUNDAMENTALS, VOL.E98–A, NO.12 DECEMBER 2015

PAPER

Data-Transfer-Aware Design of an FPGA-Based Heterogeneous Multicore Platform with Custom Accelerators

Yasuhiro TAKEI[†], ***Nonmember***, **Hasitha Muthumala WAIDYASOORIYA**[†a)], **Masanori HARIYAMA**[†], ***Members***, ***and*** **Michitaka KAMEYAMA**[†], ***Fellow***

SUMMARY For an FPGA-based heterogeneous multicore platform, we present the design methodology to reduce the total processing time considering data-transfer. The reconfigurability of recent FPGAs with hard CPU cores allows us to realize a single-chip heterogeneous processor optimized for a given application. The major problem in designing such heterogeneous processors is data-transfer between CPU cores and accelerator cores. The total processing time with data-transfers is modeled considering the overlap of computation time and data-transfer time, and optimal design parameters are searched for.

key words: *heterogeneous multicore, FPGA, custom accelerators, reconfigurable architecture*

1. Introduction

Applications used in low-power embedded processing to high performance computing have different tasks such as data-intensive tasks and control-intensive tasks. Therefore, the optimal architecture is different from application to application. Heterogeneous multicore architectures are one promising way to execute such applications power-efficiently. They use different processor cores such as CPU cores and accelerator cores as shown in Fig. 1. Examples of such processors are [1] and [2], which contain multiple CPU cores and accelerator cores. The CPU cores are suitable for control-intensive and complex computations, while the accelerator cores for data-intensive and regular computations. When tasks of an application are allocated to the most appropriate processor cores, all the cores work together to increase the overall performances power-efficiently.

Current heterogeneous processors have a fixed amount of cores and each core has a fixed amount of processing elements (PEs). Since there are many different applications, some applications may work well in a particular heterogeneous processor, while some applications may not. Moreover, large data transfer time between multiple cores is a serious problem. To solve these problems, we consider an FPGA-based heterogeneous multicore architecture model. Recently, speed and power consumption of FPGAs are greatly improved, and it would be very practical to use the FPGA-based platform for real applications. FPGAs also contain hard CPU cores as seen in Xilinx Zynq-7000 [3] and Altera Cyclone V SoC [4]. Therefore, CPU cores and accelerator cores can be efficiently implemented on a single FPGA. Moreover, recent FPGAs are large enough to hold hundreds of processor cores. Our earlier work in [5], we have proposed an FPGA-based heterogeneous multicore processor platform. However, the data transfer time between the CPU core and accelerator cores is significantly high.

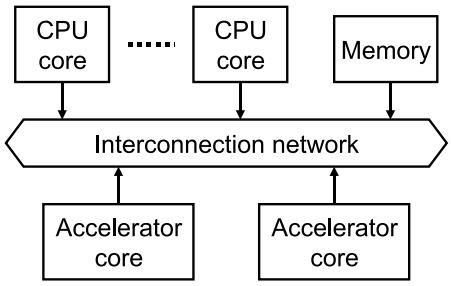


Fig. 1 Architecture of a heterogeneous multicore processor.

One popular method to reduce the data transfer time is called double buffering, where two data buffers are used. When one buffer is accessed for the computation, the data are transferred to the other buffer. After the computation is finished, the buffers are interchanged. However, this requires a large memory and only 50% is used for the computation. Since the internal (on-chip) memory is a scarce resource in FPGA, it is desirable to use most of the memory resources for the computation. Another method to reduce the data transfer time is to hide the data transfer between one core with the computations of the other cores. Since the recent FPGA-based processors contain many accelerator cores due to the large number of LUTs, we use this method to reduce the data transfer time. Since FPGA is reconfigurable, we can design the optimal architecture for different applications to reduce the processing time.

However, designing the optimal architecture that has the smallest processing time for a given application is a difficult problem and it takes a large design time. To solve this problem, we propose a very basic architecture model that has hard CPU cores and FPGA-based accelerator cores. The architecture model is based on our previous work [5]. Unlike in [5], the proposed architecture model does not contain a fixed number of accelerator cores, PEs or a fixed amount of internal memory modules. Instead we defined some design parameters such as the number of cores, the degree of parallelism, etc. We optimize our architecture model for a given application by choosing the optimal design parame-

Manuscript received March 6, 2015.
Manuscript revised June 18, 2015.
[†]The authors are with Tohoku University, Sendai-shi, 980-8579 Japan.
a) E-mail: hasitha@ecei.tohoku.ac.jp
DOI: 10.1587/transfun.E98.A.2658

ters. The optimal number of accelerator cores are chosen to hide the data transfer overhead.

In this paper, we propose a heterogeneous multicore processor design methodology to reduce the total processing time under the resource constraint. We propose a parameterized architecture model and introduce an evaluation methodology to find the optimal architecture for the design parameters. In the optimization problem, we focus on window-based processing which has many applications such as stereo matching [6], feature detection [7], scale-invariant feature transformation (SIFT) [8], histogram of oriented gradients (HOG) [9], matrix processing, filtering, etc. The evaluation using filter computation as an example demonstrates that the processing time estimated by the proposed design methodology has sufficient accuracy compared to the actual measurement of the FPGA architecture. Moreover, the optimal architecture changes for different applications, and it is possible to derive such architectures using the proposed method.

2. Heterogeneous Multicore Architecture Model

The heterogeneous multicore architecture model is based on the proposal in [5]. Figure 2 shows the overall architecture. It consists of FPGA-based custom accelerator cores, a hard CPU core and an on-chip memory. An external memory is connected to the CPU core through the FPGA board. The accelerator architecture used in this paper is based on the FE-GA (flexible engine/generic ALU array) accelerator proposed in [1]. FE-GA is a 16-bit coarse grain MIMD accelerator. It has a very simple architecture and simple interconnection network. Since the interconnection network is a critical part in FPGA-based designs, FE-GA based MIMD architecture is ideal for FPGAs. It is very easy to implement and easily scalable by changing the number of PEs and memories. Moreover, it has been studied extensively for memory allocation [10], data transfers [11], context partitioning [12], etc and many efficient techniques are proposed. It is already been used to implement various applications in many prior works, such as audio encoding [1], feature extraction [13], optical-flow extraction [14], etc. Therefore, we choose FE-GA as a base for the MIMD accelerator used in the proposed design.

Figure 3 shows the architecture of an MIMD accelerator core. It consists of a 2-dimensional array of PEs, local memory modules and address generation units (AGUs). In order to simplify the interconnection network, only the leftmost PEs can directly retrieve data from local memory modules, and only the rightmost PEs can directly write data to local memory modules. PEs, AGUs and interconnection network are dynamically reconfigurable. A PE consists of a 16-bit fixed-point ALU and a multiplier as shown in Fig. 4. It is capable of doing operations such as addition, accumulation, subtraction, comparison, absolute difference computation, multiplication, etc. Since the data path is fully pipelined, it takes only one clock cycle to complete any operation.

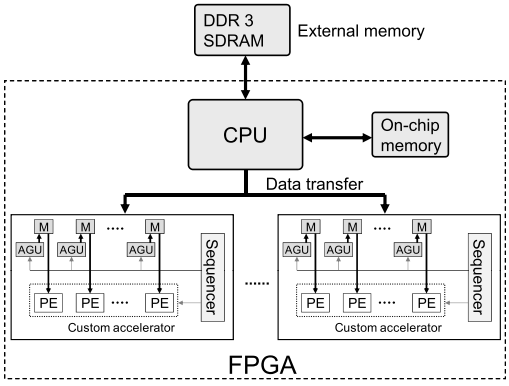


Fig. 2 Proposed heterogeneous multi-core architecture.

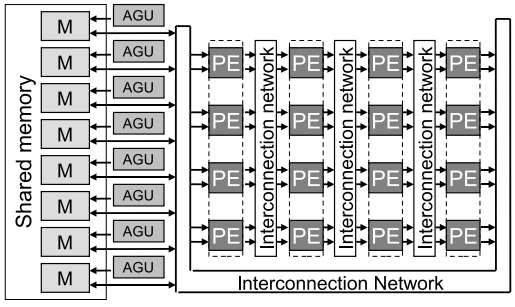


Fig. 3 MIMD accelerator architecture.

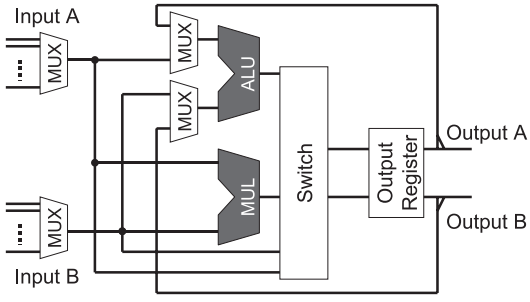
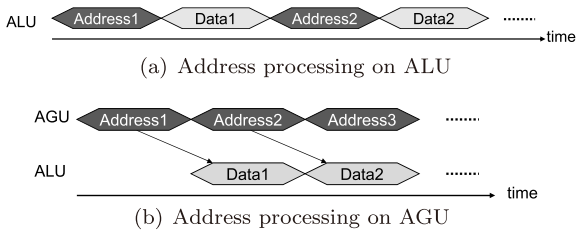


Fig. 4 Architecture of a PE.

The address calculation in the proposed architecture is explained in Fig. 5. In CPUs and GPUs, the address calculation and data processing are done on the same ALU as shown in Fig. 5(a). To reduce the address processing time, AGUs (address processing uints) are employed. The address calculation is done on AGUs in parallel to the data processing which is done on ALUs as shown in Fig. 5(b). Several address patterns and AGU architectures for image processing have been discussed in previous works [15], [16]. Since accelerator cores use multiple AGUs to access multiple memory modules, a relatively large area is required. However, considering the benefits of power-efficiency and high performance, it is worth spending resources on AGUs. In this work, the address function proposed in previous works [7], [10] is used. This address function is simple, and the resource usage of AGUs is small.

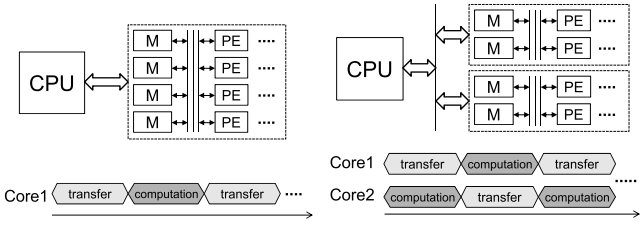
To increase the performance of an FPGA-based hetero-



(a) Address processing on ALU

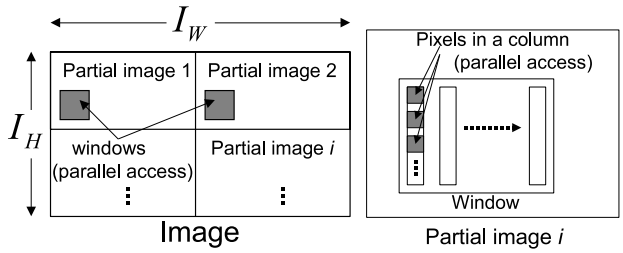
(b) Address processing on AGU

Fig. 5 Timechart of the Address processing.



(a) One core (b) Two cores

Fig. 6 Number of accelerator cores and timechart.



(a) Parallel processing of the partial images (b) Pixel-parallel column-serial processing of a window

Fig. 7 Image partition.

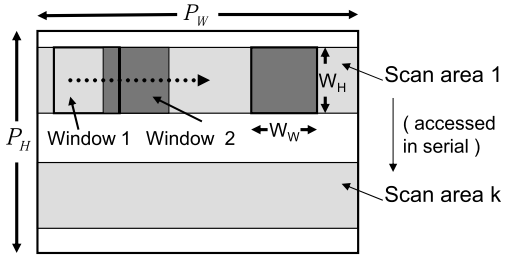


Fig. 8 Window access in a partial image.

geneous multicore platform, it is important to consider not only the architecture of accelerator core but also the number of accelerator cores. As shown in Fig. 6(a), if the number of accelerator cores is one, data-transfer and computation on an accelerator core are done in serial. On the other hands, if the number of accelerator cores is two as shown in Fig. 6(b), data-transfer to core 1 and computation on core 2 are done in parallel. Therefore, we can reduce the total processing time by changing the numbers of cores and PEs per core while keeping the total area of all cores is a constant. This techniques is used in many multicore processors such as GPUs [17], [18] and the Cell.B.E processor [19]. Since the number of cores and PEs per core are fixed in these processors, the advantages of this technique are limited. On the other hand, an FPGA-based architecture can reduce the total processing time efficiently by choosing the optimal number of cores and PEs to hide most of the data transfer time.

3. Total Processing Time Minimization

3.1 Window-Based Processing Model

We use window-based processing as an example to minimize the total processing time of an FPGA-based multicore platform. Window-based processing contains repeated access to the same data that belong to multiple overlapping windows. Therefore, it is important to maximize the data sharing, while allowing parallel processing. Such a data sharing and scheduling scheme is proposed in [10], and we use it on the proposed architecture. The work in [10] proposes an off-line scheduling scheme where a part of the data are transferred to the accelerator core, and the computation is performed. During the computation, the data are not transferred to the accelerator core. Similarly, during a data transfer, the accelerator core pauses its computations. The transferred data are stored in multiple local memory modules in the accelerator core in such a way that the data are accessed in parallel. Therefore, no data collision occurs inside an accelerator core. Please refer [10] for detailed discussions on how the data access is done inside an accelerator core. In this paper, we generalize this off-line scheduling scheme for multiple accelerator cores. The data transfer to one accelerator core starts only after the data transfers to all the other cores are finished. Therefore, no data collision occurs between the data transfer from the CPU core to the accelerator cores. Similar to [10], no data collision occurs inside an accelerator core as well.

Figure 7 shows the window-based processing model proposed in [10]. As shown in Fig. 7(a), an image is divided in to $N_{partial}$ partial images. The width and the height of the partial image is given by P_W and P_H respectively. The data between different partial images are not shared. A batch of W_P partial images are processed in parallel. The term W_P is called the degree of window parallelism and $N_{partial} \geq W_P$. The pixels in a window are accessed in pixel-parallel column-serial manner as shown in Fig. 7(b). The data in a column are accessed in parallel. This parallelism is called the pixel-parallelism and denoted by P_P.

As shown in Fig. 8 a partial image contains multiple scan areas. After the first scan area is accessed the next scan area, which is one pixel bellow, is accessed. The data in scan areas are accessed by sliding a window from left-to-right.

The different scan areas of a partial image are processed sequentially in the accelerator cores as shown in Fig. 9. The processing of a scan area is assigned to a sequence. In the first sequence, all the pixel data belong to the scan area one are transferred. In the second sequence, only the difference of the first and second scan areas is trans-

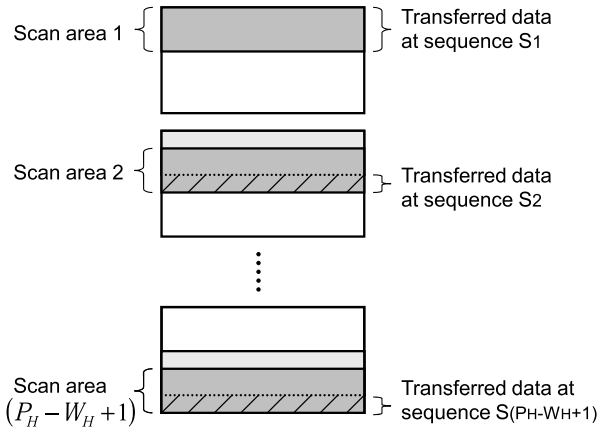


Fig. 9 Sequential processing of the scan areas.

ferred. The rest of the data are shared. Moreover, the new data are overwritten to the memory addresses with obsolete data which are not required for further processing. This method minimizes the data-transfer time since there is no data-duplication, and also optimizes the memory capacity. The width and the height of a scan area are equal to the partial image width P_W and the window height W_H respectively. Therefore, in one partial image, there are $S_{(P_H - W_H + 1)}$ scan areas.

In the above explained window-based processing model, the following relationships exists. The degree of pixel-parallelism (P_P) must satisfy the relationship given by Eq. (1), where, C_M is a natural number and W_H is the window height.

$$P_P \times C_M = W_H \tag{1}$$

The number of accelerator cores (N_C) and the degree of window-parallelism (W_P) must satisfy the relationship given by Eq. (2), where, N_W is the number of windows processed in parallel in one accelerator core.

$$W_P = N_C \times N_W \tag{2}$$

The parallelism of operations are constrained by the resources available in the FPGA. The MIMD architecture model explained in Fig. 3 contains columns of PEs where each column has n PEs. Only the first column is connected to the memory while the rest of the columns use the computation results of their previous columns. If we consider direct mapping, DFGs of most window-based applications have a tree-like structure. To implement this structure, we need $n \times (log_2 n + 1)$ number of PEs, where n and $(log_2 n + 1)$ are correspond to the depth and height of the tree. Therefore, we can have $(log_2 n + 1)$ columns of PEs in the MIMD architecture. This kind of architecture models proposed in many works such as [1]. In window-based processing, the pixel parallelism P_P equals to the number of PEs in the first column. Therefore, the number of PEs required to process at the degree of pixel parallelism equals to $P_P \times (log_2(P_P) + 1)$. Since W_P partial images are processed in parallel, the resource constraint is given by Eq. (3), where maximum number of PEs available is PE_{MAX}.

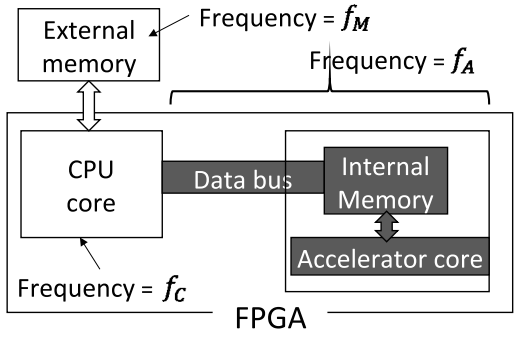


Fig. 10 Data transfer between CPU cores and accelerator cores.

$$W_P \times P_P \times (log_2 P_P + 1) \leq PE_{max} \tag{3}$$

Moreover, to implement window parallelism, the scan areas of W_P partial images must be stored in the FPGA internal memory. This constraint is given in Eq. (4)

$$W_P \times (W_H \times P_W) \leq IM_{max} \tag{4}$$

where, IM_{max} is the maximum amount of internal memory in the FPGA. Form Eqs. (3) and (4), we can see that the degree of parallelism $W_P \times P_P$ is limited by the amount of PEs and internal memory of the FPGA.

3.2 Processing Time Estimation

In this section, we explain the formulation of the processing time minimization problem. The processing of each sequence (a scan area) in Fig. 9 is divided into the following three phases.

Phase 1 Data-transfer from the CPU cores to the accelerator cores
Phase 2 Computation on the accelerator cores
Phase 3 Data-transfer from the accelerator cores to the CPU cores.

In this section, we discuss the processing time estimation of each phase and the total processing time.

3.2.1 Data-Transfer Time from CPU Cores to Accelerator Cores (Phase 1)

The data are transferred from the CPU core to the accelerator core through the AXI (Advanced eXtensible Interface) bus connected to the ARM processor. The bus width of the accelerator core's input memory and the word width of the input data are given by B_B and B_{CA}. If $B_B \geq B_{CA}$, we can transfer several data in parallel. On the other hand, if $B_B < B_{CA}$, one word is divided in to several segments and transfer each segment in a serial manner. Therefore, the number of words transferred from CPU core to accelerator core at a time (N_{CA}) is given by Eq. (5).

$$N_{CA} = \begin{cases} \left\lfloor \frac{B_B}{B_{CA}} \right\rfloor & \text{when } B_B \geq B_{CA} \\ 1 / \left\lceil \frac{B_{CA}}{B_B} \right\rceil & \text{when } B_B < B_{CA} \end{cases} \tag{5}$$

The data transfer between CPU cores and accelerator cores is shown in Fig. 10. The frequencies of the external memory, CPU cores and accelerator cores may not be the same. Even though frequencies does not match, CPU cores and the data bus has necessary hardware such as memory controllers, data buffers, etc for an efficient data transfer. However, we cannot determine the transfer speed by using the parameters such as bus width, frequencies, etc. Therefore, we measure the data transfer time between CPU and accelerator cores using sample data. The term α is the average time per word-transfer from CPU cores to accelerator cores.

The amount of words transferred in sequence S_1 is different from those in the other sequences. In sequence S_1, the amount of words transferred from a CPU core to an accelerator core is $P_W \times W_H$ as shown in Fig. 9. Data-transfer time from a CPU core to an accelerator core in the sequence $S_1(t_{CA1})$ is given by

$$t_{CA1} = \alpha \times \left\lceil \frac{N_W}{N_{CA}} \right\rceil \times P_W \times W_H \tag{6}$$

Note that N_W is the number of windows processed in an accelerator core as described in Eq. (2). In other sequence (S_2 to $S_{(P_H - W_H + 1)}$), the amount of words transferred from a CPU core to an accelerator core is P_W. Data-transfer time from a CPU core to an accelerator core in each sequence (t_{CA2}) is given by

$$t_{CA2} = \alpha \times \left\lceil \frac{N_W}{N_{CA}} \right\rceil \times P_W. \tag{7}$$

3.2.2 Computation Time (Phase 2)

We estimate the computation time (t_{comp}) in each sequence on the accelerator core. The architecture of the accelerator is fully pipelined. After the pipeline is filled, the computation is done in every clock cycles. As shown in Fig. 8, the size of the window is $W_H \times W_W$. Each scan area includes $(P_W - W_W + 1)$ windows. When processing a window, P_P pixels are calculated in parallel as shown in Fig. 7(b). Therefore, t_{comp} is given by

$$t_{comp} = \frac{1}{f_A} \times \frac{W_H \times W_W}{P_P} \times (P_W - W_W + 1) + t_{pipe} \tag{8}$$

where f_A is the clock frequency, and t_{pipe} is the pipeline latency of the accelerator core. Note that the address generation time does not appear in Eq. (8) since the address processing time is completely overlapped with the data processing time. This is because the address and data processing are done in AGUs and PEs respectively in parallel.

3.2.3 Data-Transfer Time from Accelerator Cores to CPU Cores (Phase 3)

We estimate the data-transfer time (t_{AC}) from accelerator cores to CPU cores in each sequence by the same method described in Sect. 3.2.1. The number of words transferred from accelerator cores to CPU cores at a time (N_{AC}) is given by Eq. (9) where, B_{AC} is the word width of the output data of the accelerator cores.

$$N_{AC} = \begin{cases} \left\lfloor \frac{B_B}{B_{AC}} \right\rfloor & \text{when } B_B \geq B_{AC} \\ 1/\left\lceil \frac{B_{AC}}{B_B} \right\rceil & \text{when } B_B < B_{AC} \end{cases} \tag{9}$$

The amount of words transferred from an accelerator core to a CPU core is $(P_W - W_W + 1)$ in each sequence. Data-transfer time from an accelerator core to a CPU core in each sequence (t_{AC}) is given by

$$t_{AC} = \beta \times \left\lceil \frac{N_W}{N_{AC}} \right\rceil \times (P_W - W_W + 1) \tag{10}$$

where β is the average time per word-transfer from accelerator core to CPU cores.

3.2.4 Estimation of the Total Processing Time

We process W_P partial images in parallel, and the processing time required for this is given by $t_{partial}$. Figure 11 shows the time chart of the processing in an accelerator core. The time t_{init} consists of the initial data-transfer time from the CPU cores to the accelerator cores and the computation time in the sequence S_1. These initial data-transfers to different cores cannot be done in parallel since there is only one bus that has a limited bandwidth. Therefore, t_{init} is given by

$$t_{init} = t_{CA1} \times N_C + t_{comp} \tag{11}$$

During t_{mid}, t_{trans} and t_{comp} are repeated as shown in Fig. 11. The time t_{trans} shown in Fig. 11 is defined by Eq. (12)

$$t_{trans} = t_{AC} + t_{CA2} + t_{ctrl} \tag{12}$$

where, t_{AC} is the data-transfer time from accelerator cores to CPU core, t_{CA2} is the data-transfer time from CPU cores to accelerator cores and t_{ctrl} is the control overhead due to starting and stopping the accelerator cores. To estimate t_{mid}, we have to consider the overlap between the data-transfers and the computations. This overlap can be classified into following two cases.

Case A1: The computation time of one core partially hides the data-transfers of the other cores as shown in Fig. 12(a). This is represented by $t_{comp} < (N_C - 1) \times t_{trans}$. During t_{mid}, the time period $(N_C \times t_{trans})$ is repeated $(P_H - W_H)$ times since there are $P_H - W_H + 1$ sequences as explained in Sect. 3.1 and $(P_H - W_H)$ of those sequences belong to t_{mid}.

Case A2: The computation time completely hides the data-transfer time of the other cores as shown in Fig. 12(b). This is represented by $t_{comp} \geq (N_C - 1) \times t_{trans}$. Similar to the case A1, the time period $(t_{comp} + t_{trans})$ is repeated $(P_H - W_H)$ times during t_{mid}. According to these two types, t_{mid} is given by Eq. (13).

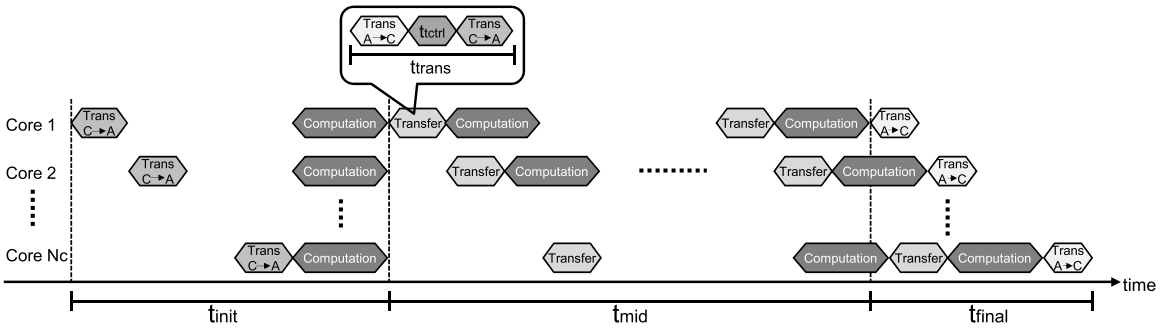
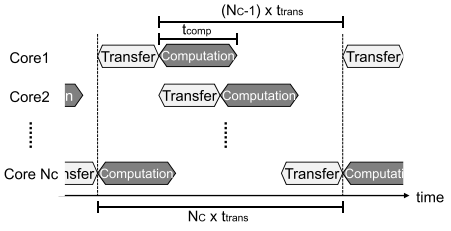
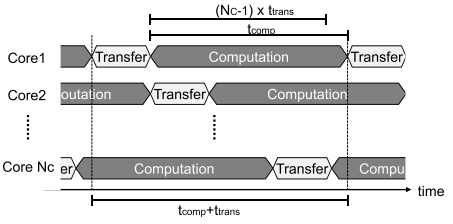


Fig. 11 Timechart of the processing in the accelerator cores.



(a) Case A1: The data-transfers to cores 2∼Nc are not hidden completely by computation on core 1



(b) Case A2: The data-transfers to cores 2∼Nc is hidden completely by computation on core 1

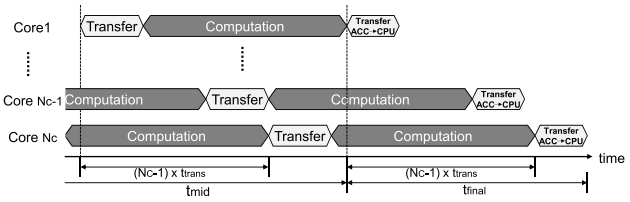
Fig. 12 The data-transfers and the computations during t_{mid}

$$t_{mid} = \begin{cases} N_C \times t_{trans} \times (P_H - W_H) \\ \text{when } \ t_{comp} < (N_C - 1) \times t_{trans} \\ \qquad\qquad\qquad\qquad \text{(Case1 A1)} \\ (t_{trans} + t_{comp}) \times (P_H - W_H) \\ \text{when } \ t_{comp} \geq (N_C - 1) \times t_{trans} \\ \qquad\qquad\qquad\qquad \text{(Case A2)} \end{cases} \tag{13}$$

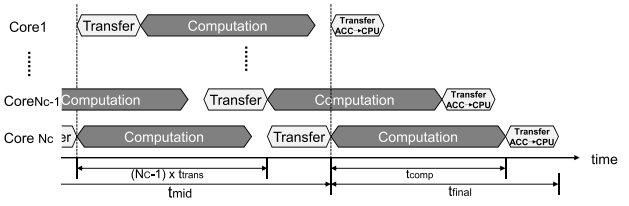
The time t_{final} is the data-transfer time from the accelerator cores to the CPU cores in the sequence $S_{(P_H - W_H + 1)}$. During t_{final}, the data-transfers form the accelerator cores to the CPU cores (t_{AC}) overlap with the computations as shown in Fig. 13. This can be classified into three cases.

Case B1 $t_{comp} \geq (N_C - 1) \times t_{trans}$
Case B2 $(N_C - 1) \times t_{AC} \leq t_{comp} < (N_C - 1) \times t_{trans}$
Case B3 $t_{comp} < (N_C - 1) \times t_{AC}$

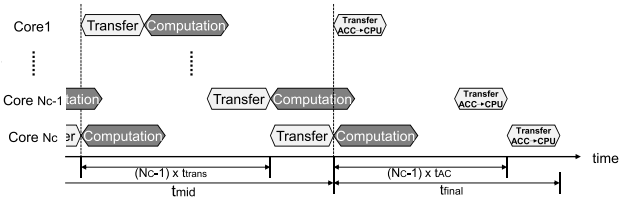
In case B1, t_{AC} is smaller than t_{trans}, so that t_{comp} of one core hides t_{AC} of all the other cores as shown in Fig. 13(a). When $t_{comp} < (N_C - 1) \times t_{trans}$, t_{comp} can be either "greater than or equals to $(N_C - 1) \times t_{AC}$" or "smaller than $(N_C - 1) \times t_{AC}$". In case 2, t_{comp} of one core completely hides the t_{AC} of the other cores as shown in Fig. 13(b). In case 3, t_{comp} of one core partially hides the t_{AC} of the other cores as shown in Fig. 13(c). According to these three cases, t_{final} is given by Eq. (14).



(a) Case B1: $t_{comp} \geq (N_C - 1) \times t_{trans}$



(b) Case B2: $(N_C - 1) \times t_{AC} \leq t_{comp} < (N_C - 1) \times t_{trans}$



(c) Case B3: $t_{comp} \leq (N_C - 1) \times t_{AC}$

Fig. 13 Overlap with data-transfer and computation in t_{final}.

$$t_{final} = \begin{cases} (N_C - 1) \times t_{trans} + t_{AC} \\ \text{when } t_{comp} \geq (N_C - 1) \times t_{trans} \\ \qquad\qquad\qquad\qquad \text{(Case B1)} \\ t_{AC} + t_{comp} \\ \text{when } (N_C - 1) \times t_{AC} \leq t_{comp} \\ \qquad < (N_C - 1) \times t_{trans} \\ \qquad\qquad\qquad\qquad \text{(Case B2)} \\ N_C \times t_{AC} \\ \text{when } t_{comp} < (N_C - 1) \times t_{AC} \\ \qquad\qquad\qquad\qquad \text{(Case B3)} \end{cases} \tag{14}$$

The processing time required for W_P partial images ($t_{partial}$) is given by

$$t_{partial} = t_{init} + t_{mid} + t_{final} \quad (15)$$

The total processing time denoted by T_{image} is the time required to process a whole image. As explained in Sect. 3.1, an image is divided into $N_{partial}$ partial images, and W_P partial images are processed in parallel. Equation (15) gives the processing time of W_P partial images. After processing W_P partial images, another W_P partial images are processed. Therefore the total processing time required to process a whole image is given by

$$T_{image} = t_{partial} \times \left\lceil \frac{N_{partial}}{W_P} \right\rceil \quad (16)$$

Note that, for smaller images where $N_{partial} = W_P$, T_{image} equals $t_{partial}$.

Using Eqs. (11), (13), (14) and (15), we can see that the total processing time is derived from the combination of the design parameters, W_P, P_P, N_C, N_W, P_W and P_H. These parameters define the architecture of the FPGA-based heterogeneous multicore platform and its scheduling. Therefore, it is very important to find the optimal combination of design parameters that minimize the total processing time. The design parameter optimization is discussed in Sect. 4.

4. Evaluation

We use Xilinx Zynq-7000 EPP ZC702 board [20] for the evaluation. Zynq integrates a dual Cortex-A9 CPU cores and FPGA equivalent to Atrix-7 on a single chip. In addition, the evaluation kit has a DDR3 SDRAM for an external memory. The proposed heterogeneous platform is designed using Xilinx PlanAhead 14.2. The CPU core is programmed using C language on Xilinx EDK 14.2. Figure 14 shows the implemented architecture. There are accelerator cores, one Cortex-A9 hard CPU core, the AXI4 bus and a DDR3 SDRAM for the external memory. The processing time of the proposed heterogeneous platform is measured by the AXI Timer IP. The clock frequency of the CPU core is 667 MHz.

To estimate the data-transfer time of the proposed heterogeneous platform given by Eq. (15), we measured the values α, β and t_{ctrl} in Eqs. (6), (10) and (12) respectively. We measured data-transfer times between the external memory and memory modules of accelerator cores. From experimental results, the values of α, β and t_{ctrl} are measured to be 186.06 (ns), 213.02 (ns) and 430.00 (ns) respectively, the maximum clock frequency of the accelerator cores is measured to be 100 MHz. Table 1 shows the difference between the estimated processing time and the measured processing time for different windows sizes. The estimated processing time is calculated using Eq. (15) for a given set of design parameters. We implement the architecture described by the same set of parameters on FPGA and measure the processing time. This is called the measured processing time. According to the results, the error rate calculated using Eq. (17) is less than 1%. This small error percentage shows that the estimated processing time is sufficiently accurate to optimize the processor architecture.

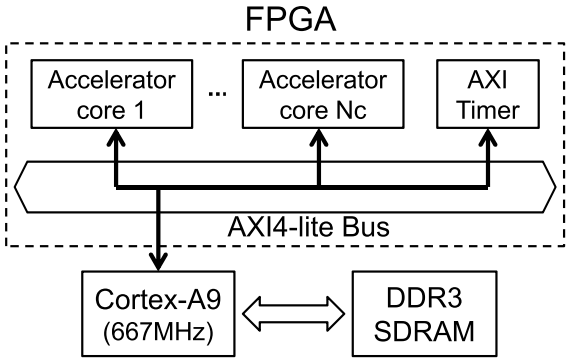


Fig. 14 Implemented architecture.

Table 1 Estimated time vs. measured time.

Window size	Estimated time(ms)	Measured time(ms)	Error (%)
12×12	46.93	46.51	0.51
16×16	60.38	60.07	0.91
18×18	70.50	70.51	0.02
24×24	115.89	115.87	0.01

Image width (I_W)	640 pixels
Image height (I_H)	480 pixels
Window width (W_W)	16
Window height (W_H)	16
Maximum parallelism	16
Bus width (B_B)	32
Word width (B_{CA})	8
Word width (B_{AC})	16

(a) Specifications of the filter computation

Design parameter	Scope	
	from	to
Partial image width (P_W)	16	640
Partial image height (P_H)	16	480
Window parallelism (W_P)	1	16
Pixel parallelism (P_P)	1	16
Number of cores (N_C)	1	16
Windows per a core (N_W)	1	16

(b) Scope of the design parameters

Fig. 15 Filter computation of a VGA image.

$$\text{Error} = \frac{|\text{Measured} - \text{Estimated}|}{\text{Measured}} \times 100(\%) \quad (17)$$

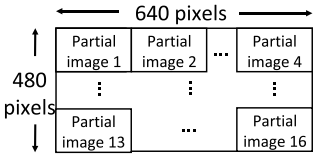
4.1 Exploration of the Design Parameter Space to Find the Minimum Total Processing Time

In this section, we show how the design parameter space is explored to obtain the optimal ones that give the minimum processing time for filter computation. The specifications of the filter computation are given in Fig. 15. We assume that the maximum degree of parallelism ($W_P \times P_P$) is limited to 16 by the resource constraints in Eqs. (3) and (4). Based on the specifications, the scope of the design parameters are determined as shown in Fig. 15(b). We estimate the total processing time for all the combinations of the design parameters in order to find the optimal parameters that gives the minimum processing time.

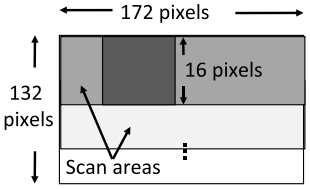
Figure 16 shows an example of problem formulation for a given set of design parameters. The design parameters

Partial image width (P_W)	172 pixels
Partial image height (P_H)	132 pixels
Window parallelism (W_P)	16
Pixel parallelism (P_P)	1
Number of cores (N_C)	4
Windows per a core (N_W)	4

(a) Design parameter combination



(b) Image partitioning



(c) Scan area of a partitial image

Fig. 16 Problem formulation for a given set of design parameters.

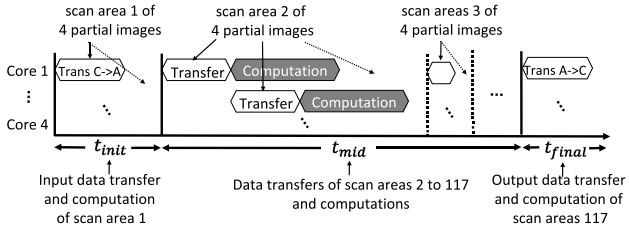


Fig. 17 Processing time estimation.

are shown in Fig. 16(a). According to the parameters P_W and P_H, we partition the image into 16 partial images as shown in Fig. 16(b). Since the number of cores (N_C) is 4, we assign four different partial images to each core. The scan areas of a partial image is shown in Fig. 16(c). Since $N_W = 4$, one core processes four scan areas belongs to four different partial images in parallel.

Figure 17 shows the time chart of processing. As explained in Sect. 3.2, the total processing time is the summation of t_{init}, t_{mid} and t_{total}. Each term is calculated using Eqs. (11), (13) and (14) respectively. During t_{init}, the first scan area belongs to a partial image is transferred to the accelerator cores. Since the bus-width (B_B) is 32 bits and the input data width B_{CA} is 8 bits, The data of four windows are transferred to the accelerator core in parallel. However, the output data width B_{AC} is 16 bits so that only the processing results of only two windows are transferred in parallel during t_{mid} and t_{final}. During t_{mid}, the data transfers and the computations of scan areas 2 to 117 are done. During t_{final}, the remaining computations and the output data transfer corresponds to the last scan area is done.

We estimated the total processing time for all combinations of design parameters by doing an exhaustive search. The search could be done in few minutes on an Intel CPU at 3.2 GHz. Table 2 shows the processing time for some of those combinations. The total processing time is minimized when $W_P = 16$, $P_P = 1$, $N_C = 4$, $N_W = 4$, $P_W = 94$ and $P_H = 246$. Usually, it would take several days of designing and compilation time to design an FPGA architecture. In the proposed method, we can design a reasonably good FPGA-based heterogeneous processor architecture by just searching for the optimal design parameters. Even using exhaustive search, the optimal design parameters are found in a very short time. Therefore, we can reduce the FPGA architecture design effort and time dramatically by the proposed method. When exploring the design parameter space, we consider all possible situations that belongs two different cases shown in Fig. 12. For example, in the last column of Table 2, $t_{comp} = 0.4020$ and $(N_C - 1) \times t_{trans} = 0.994$. Therefore, $t_{comp} < (N_C - 1) \times t_{trans}$ so that this design belongs to case A1, which is shown in Fig. 12(a).

4.2 Evaluation of the Optimized Design for Different Filter Sizes

Table 3 shows the optimized design parameters for different filter sizes. Note that the window size equals to the filter size. We assume that the maximum degree of parallelism is 16 due to the resource constraints. According to the results, optimized design parameters vary with different filter sizes. That means, the partitioning, scheduling and the hardware design are different according to the specifications of the application. However, in conventional heterogeneous multicore architectures such as [1], we cannot optimize the design parameters since the number of accelerator cores, the number PEs, etc are fixed. Therefore, the proposed FPGA-based heterogeneous platform has a high degree of flexibility, and can be optimized for different applications.

The optimized design (decided by the design parameters) in Table 3 belongs to either the case A1 in Fig. 12(a) or the case A2 in Fig. 12(b). This shows that considering both cases is important to obtain the optimal design. Although not shown in Table 3, each of the above cases are further divided in to another three cases, B1~B3 as shown in Fig. 13. When exploring the design parameter space, we considered these cases also.

Tables 4 and 5 shows the optimized design parameters when the maximum degree of parallelism is 32 and 64 respectively. When the degree of parallelism increases, the computation time reduces. As a result, the data transfer time could be larger than the computation time. Therefore, most of the optimized designs belong to the case A1 where the data transfer time is not fully hidden by the computation time.

For some optimized designs, the degree of parallelism is smaller than the maximum value. For example, when the window size is 8×8 in Table 5, the maximum degree of parallelism allowed is 64. However, the degree of window and pixel parallelisms in the optimized design are 8 and 2 respectively. This give a total degree of parallelism of 16 which is just 25% of the maximum available. In this case, the computation time is much smaller than the data transfer time due to the parallel computations. Therefore, the total processing time is decided by the data transfer time as shown in Fig. 12(a), so that reducing the computation time further does not make any impact on the total processing

Table 2 Estimated processing time for different design parameters. All different combinations of design parameters are searched to find the optimal parameters that minimize the total processing time.

Design Parameters						Type of the design			Total processing time (ms)
Degree of window parallelism W_P	Degree of pixel parallelism P_P	Number of cores N_C	Windows per a core N_W	Partial image width P_W	Partial image height P_H	t_{comp} (ms)	t_{trans} (ms)	Type	
1	4	1	1	640	480	0.4001	0.2531	Case A2	305.80
4	2	1	4	328	248	0.4007	0.1868	Case A2	137.98
4	2	4	1	172	480	0.2010	0.0663	Case A2	127.02
4	2	2	2	172	480	0.2010	0.0663	Case A2	125.61
16	1	1	16	94	248	0.2024	0.1981	Case A2	94.70
16	1	2	8	94	248	0.2023	0.0993	Case A2	71.83
16	1	4	4	172	132	0.4020	0.0955	Case A2	61.02
16	1	4	4	94	248	0.2023	0.0499	Case A2	60.38*
16	1	8	2	94	248	0.4020	0.0663	Case A1	67.47
16	1	16	1	172	132	0.4020	0.0663	Case A1	134.14

*The smallest total processing time.

Table 3 Optimized design parameters for different filter sizes. The maximum degree of parallelism is 16.

Window size $W_H = W_W$	Design Parameters						Total processing time (ms)	Type of the design
	Window parallelism W_P	Pixel parallelism P_P	Number of cores N_C	Windows per a core N_W	Partial image width P_W	Partial image height P_H		
8	16	1	2	8	166	126	46.24	Case A1
9	16	1	2	8	166	126	45.88	Case A1
10	16	1	2	8	167	127	46.20	Case A1
12	16	1	4	4	169	129	46.93	Case A1
13	16	1	4	4	169	129	46.62	Case A1
15	16	1	4	4	93	247	54.50	Case A2
16	16	1	4	4	94	248	60.38	Case A2
17	16	1	4	4	94	248	65.53	Case A2
18	16	1	8	2	95	249	70.50	Case A2
20	16	1	8	2	58	480	83.97	Case A2
24	16	1	8	2	62	480	115.89	Case A2

Table 4 Optimized design parameters for different filter sizes. The maximum degree of parallelism is 32.

Window size $W_H = W_W$	Design Parameters						Total processing time (ms)	Type of the design
	Window parallelism W_P	Pixel parallelism P_P	Number of cores N_C	Windows per a core N_W	Partial image width P_W	Partial image height P_H		
8	8	2	2	4	166	244	45.58	Case A1
9	16	1	2	8	166	126	45.88	Case A1
10	8	2	2	4	167	245	45.53	Case A1
12	16	1	4	4	169	129	46.93	Case A1
13	16	1	4	4	169	129	46.62	Case A1
15	32	1	2	16	93	131	49.00	Case A1
16	32	1	4	8	94	132	49.53	Case A1
17	32	1	4	8	94	132	49.05	Case A1
18	32	1	4	8	95	133	49.55	Case A1
20	32	1	8	4	97	135	50.75	Case A1
24	32	1	8	4	62	252	61.60	Case A2

time.

Table 6 shows the comparison of the total processing time of the proposed method against the method given in [10]. The window-based processing model used in both methods are the same. However, the work in [10] does not consider the overlap of data transfers with the computation. In [10], the total processing time is calculated simply by adding the total computation time, the total data transfer time and the total control time together. It uses one large accelerator core that can process the data in parallel using multiple PEs. The comparison is done for three different resource constraints. The maximum degree of parallelism ($W_P \times P_P$) is calculated to be 16, 32 and 64 in each of the three constraints.

Table 5 Optimized design parameters for different filter sizes. The maximum degree of parallelism is 64.

Window size $W_H = W_W$	Design Parameters						Total processing time (ms)	Type of the design
	Window parallelism W_P	Pixel parallelism P_P	Number of cores N_C	Windows per a core N_W	Partial image width P_W	Partial image height P_H		
8	8	2	2	4	166	244	45.58	Case A1
9	16	1	2	8	166	126	45.88	Case A1
10	8	2	2	4	167	245	45.53	Case A1
12	16	2	2	8	169	129	46.93	Case A1
13	16	1	4	4	169	129	46.62	Case A1
15	32	1	2	16	93	131	49.00	Case A1
16	16	2	4	4	172	132	47.33	Case A1
17	32	1	4	8	94	132	49.05	Case A1
18	16	2	4	4	173	133	47.35	Case A1
20	16	2	4	4	97	250	49.75	Case A2
24	64	1	4	16	62	480	57.43	Case A1

Table 6 Total processing time reduction of the proposed method compared to [10].

Maximum degree of parallelism	Window size $W_H = W_W$	Work in [10]				This work	Reduction Percentage (%)
		Computation time (ms)	Control time (ms)	Data transfer time (ms)	Total processing time (ms)	Total processing time (ms)	
16	8	12.00	0.20	44.21	56.42	46.24	18.04
	10	18.60	0.20	44.01	62.81	46.20	26.45
	15	41.05	0.20	43.50	84.75	54.50	35.69
	20	71.60	0.20	43.00	114.79	83.97	26.85
	24	101.53	0.20	42.59	144.32	115.89	19.70
32	8	6.02	0.20	44.21	50.43	45.58	9.63
	10	9.32	0.20	44.01	53.53	45.53	14.94
	15	20.54	0.20	43.50	64.24	49.00	23.72
	20	35.81	0.20	43.00	79.00	50.73	35.79
	22	43.00	0.20	42.79	85.99	54.10	37.09
	24	50.78	0.20	42.59	93.57	61.60	34.17
64	8	3.02	0.20	44.21	47.44	45.58	3.92
	10	4.67	0.20	44.01	48.88	45.53	6.86
	15	10.28	0.20	43.50	53.98	49.00	9.23
	20	17.92	0.20	43.00	61.11	49.75	18.59
	24	25.40	0.20	42.59	68.19	57.43	15.78

According to the results in Table 6, the total processing time is reduced up to 37% using the proposed method. The computation amount increases with the window size. For small window sizes, the computation time is smaller compared to the data transfer time so that most of the data transfer time is not hidden. For larger window sizes, the computation time is larger compared to the data transfer time due to large computation amount. In this case, the data transfer time is hidden, so that the total processing time is decided by the computation time. Therefore, the largest total processing time reduction is achieved when the computation time nearly equals the data transfer time. When we increase the degree of parallelism, the computation time decreases. However, both the data amount and the number of parallel data transfers are unchanged (the maximum amount of parallel data transfers is already reached), so that the data transfer time remains the same. Nevertheless, the total processing time is reduced in both methods due to the reduction of the computation time.

According to the results in Table 6, the total processing time in the proposed method is smaller than that in [10]. Even in the extreme cases where one of the data transfer time or the computation time is negligibly smaller compared to the other, the total processing time of the proposed method would be at least equals to [10]. During the design parameter exploration, the method in [10] is also one of over thousand combinations considered by the proposed method. Therefore the proposed method would never get worse than that in [10].

5. Conclusion

We have proposed a design methodology for FPGA-based heterogeneous multi-core platform with custom accelerator cores. The proposed approach optimizes the total processing time by considering this overlap of data-transfers and computations. According to the evaluation, the processing time estimation has a sufficient accuracy, so that the architecture of the FPGA-based heterogeneous platform can be optimized. FDTD (finite-difference time-domain) computation, which is basically a stencil computation, has already been implemented in FPGAs [21], [22]. Therefore, we believe that the proposed architecture model could be optimized for such applications grid-based HPC (high perfor-

mance computing) applications such as stencil computation [23] in future works.

Acknowledgment

This work is supported by MEXT KAKENHI Grant Number 24300013.

References

[1] H. Shikano, M. Ito, M. Onouchi, T. Todaka, T. Tsunoda, T. Kodama, K. Uchiyama, T. Odaka, T. Kamei, E. Nagahama, M. Kusaoke, Y. Nitta, Y. Wada, K. Kimura, and H. Kasahara, "Heterogeneous multicore architecture that enables 54x AAC-LC stereo encoding," IEEE J. Solid-State Circuits, vol.43, no.4, pp.902–910, 2008.

[2] H. Kondo, M. Nakajima, N. Masui, S. Otani, N. Okumura, Y. Takata, T. Nasu, H. Takata, T. Higuchi, M. Sakugawa, H. Fujiwara, K. Ishida, K. Ishimi, S. Kaneko, T. Itoh, M. Sato, O. Yamamoto, and K. Arimoto, "Design and implementation of a configurable heterogeneous multicore SoC with nine CPUs and two matrix processors," IEEE J. Solid-State Circuits, vol.43, no.4, pp.892–901, 2008.

[3] S. Dutta, V. Rajagopolan, B. Taylor and R. Wittig, "Xilinx Zynq embedded processing platform" A Symp. High Performance Chips (HOT Chips 23), 2011.

[4] https://www.altera.com/products/soc/portfolio/cyclone-v-soc/overview.html

[5] Y. Takei, H.M. Waidyasooriya, M. Hariyama, and M. Kameyama, "Evaluation of an FPGA-based heterogeneous multicore platform with SIMD/MIMD custom accelerators," IEICE Trans. Fundamentals, vol.E96-A, no.12, pp.2576–2586, 2013.

[6] M. Hariyama, H. Sasaki, and M. Kameyama, "Architecture of a stereo matching VLSI processor based on hierarchically parallel memory access," IEICE Trans. Inf. & Syst., vol.E88-D, no.7, pp.1486–1491, 2005.

[7] H.M. Waidyasooriya, Y. Ohbayashi, M. Hariyama, and M. Kameyama, "Memory-access-driven context partitioning for window-based image processing on heterogeneous multicore processors," IEICE Trans. Inf. & Syst., vol.E95-D, no.2, pp.354–363, 2012.

[8] D.G. Lowe, "Object recognition from local scale-invariant features," Proc. Seventh IEEE International Conference on Computer Vision, vol.2, pp.1150–1157, 1999.

[9] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol.1, pp.886–893, 2005.

[10] H.M. Waidyasooriya, Y. Ohbayashi, M. Hariyama, and M. Kameyama, "Memory allocation exploiting temporal locality for reducing data-transfer bottlenecks in heterogeneous multicore processors," IEEE Trans. Circuits Syst. Video Technol., vol.21, no.10, pp.1453–1466, 2011.

[11] Y. Hiramatsu, H.M. Waidyasooriya, M. Hariyama, T. Nojiri, K. Uchiyama, and M. Kameyama, "Acceleration of block matching on a low-power heterogeneous multi-core processor based on DTU data-transfer with data re-allocation," IEICE Trans. Electron., vol.E95-C, no.12, pp.1872–1882, 2012.

[12] H.M. Waidyasooriya, Y. Ohbayashi, M. Hariyama, and M. Kameyama, "Memory-access-driven context partitioning for window-based image processing on heterogeneous multicore processors," IEICE Trans. Inf. & Syst., vol.E95-D, no.2, pp.354–363, 2012.

[13] H.M. Waidyasooriya, D. Okumura, M. Hariyama, and M. Kameyama, "Task allocation with algorithm transformation for reducing data-transfer bottlenecks in heterogeneous multi-core processors: A case study of HOG descriptor computation," IEICE Trans. Fundamentals, vol.E93-A, no.12, pp.2570–2580, 2010.

[14] H.M. Waidyasooriya, M. Hariyama, and M. Kameyama, "Acceleration of optical-flow extraction using dynamically reconfigurable ALU arrays," International Conference on Engineering of Reconfigurable Systems and Algorithms, pp.291–294, 2009.

[15] Y. Kobayashi, M. Hariyama, and M. Kameyama, "Memory allocation for multi-resolution image processing," IEICE Trans. Inf. & Syst., vol.E91-D, no.10, pp.2386–2397, 2008.

[16] H.M. Waidyasooriya, M. Hariyama, and M. Kameyama, "Memory allocation for window-based image processing on multiple memory modules with simple addressing functions," IEICE Trans. Fundamentals, vol.E94-A, no.1, pp.342–351, 2011.

[17] H.-N. Ta and S. Lee, "High-performance computing model for 3D camera system," 2011 IEEE International Conference on Robotics and Biomimetics, pp.354–359, 2011.

[18] K. Iwai, T. Kurokawa, and N. Nisikawa, "AES encryption implementation on CUDA GPU and its analysis," 2010 First International Conference on Networking and Computing, pp.209–214, 2010.

[19] T. Chen, Z. Sura, K. O'Brien and J.K. O'Brien, "Optimizing the use of static buffers for DMA on a CELL chip," Proc. 19th Int. Conf. Languages and Compilers for Parallel Computing, vol.4382, pp.314–329, 2006.

[20] http://www.xilinx.com/support/documentation/boards_and_kits/zc702_zvik/ug850-zc702-eval-bd.pdf

[21] J.P. Durbano, F.E. Ortiz, J.R. Humphrey, P.F. Curt, and D.W. Prather, "FPGA-based acceleration of the 3D finite-difference time-domain method," 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp.156–163, 2004.

[22] H.M. Waidyasooriya, Y. Takei, M. Hariyama, and M. Kameyama, "Low-power heterogeneous platform for high performance computing and its application to 2D-FDTD computation," Proc. International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), pp.147–150, 2012.

[23] K. Datta, M. Murphy, V. Volkov, S. Williams, J. Carter, L. Oliker, D. Patterson, J. Shalf, and K. Yelick, "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures," International Conference for High Performance Computing, Networking, Storage and Analysis, pp.1–12, 2008.

Yasuhiro Takei received the B.E. degree in electronic engineering in Information Sciences, M.S. degree in Information Sciences from Tohoku University, Sendai, Japan, in 2011 and 2013 respectively. He is currently a Ph.D. student in Graduate School of Information Sciences, Tohoku University. His research interests include heterogeneous multicore processor architectures.

Hasitha Muthumala Waidyasooriya received the B.E. degree in Information Engineering, M.S. degree in Information Sciences and Ph.D. in Information Sciences from Tohoku University, Japan, in 2006, 2008 and 2010 respectively. He is currently an Assistant Professor with the Graduate School of Information Sciences, Tohoku University. His research interests include reconfigurable computing, processor architectures for big-data processing and high-level design methodology for VLSIs.

Masanori Hariyama received the B.E. degree in electronic engineering, the M.S. degree in information sciences, and the Ph.D. degree in information sciences from Tohoku University, Sendai, Japan, in 1992, 1994, and 1997, respectively. He is currently an Associate Professor with the Graduate School of Information Sciences, Tohoku University. His research interests include real-world applications such as robotics and medical applications, big data applications such as bio-informatics, high-performance computing, VLSI computing for real-world application, high-level design methodology for VLSIs, and reconfigurable computing.

Michitaka Kameyama received the B.E., M.E. and D.E. degrees in Electronic Engineering from Tohoku University, Sendai, Japan, in 1973, 1975, and 1978, respectively. He is currently a Professor in the Graduate School of Information Sciences, Tohoku University. His general research interests are intelligent integrated systems for real-world applications and robotics, advanced VLSI architecture, and new-concept VLSI including multiple-valued VLSI computing. Dr. Kameyama received the Outstanding Paper Awards at the 1984, 1985, 1987 and 1989 IEEE International Symposiums on Multiple-Valued Logic, the Technically Excellent Award from the Society of Instrument and Control Engineers of Japan in 1986, the Outstanding Transactions Paper Award from the IEICE in 1989, the Technically Excellent Award from the Robotics Society of Japan in 1990, and the Special Award at the 9th LSI Design of the Year in 2002. Dr. Kameyama is an IEEE Fellow.