IEICE TRANS. FUNDAMENTALS, VOL.E96–A, NO.12 DECEMBER 2013

PAPER *Special Section on VLSI Design and CAD Algorithms*

Evaluation of an FPGA-Based Heterogeneous Multicore Platform with SIMD/MIMD Custom Accelerators

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

SUMMARY Heterogeneous multi-core architectures with CPUs and accelerators attract many attentions since they can achieve power-efficient computing in various areas from low-power embedded processing to high-performance computing. Since the optimal architecture is different from application to application, finding the most suitable accelerator is very important. In this paper, we propose an FPGA-based heterogeneous multi-core platform with custom accelerators for power-efficient computing. Using the proposed platform, we evaluate several applications and accelerators to identify many key requirements of the applications and properties of the accelerators. Such an evaluation is very important to select and optimize the most suitable accelerator according to the requirements of an application to achieve the best performance.

key words: *heterogeneous multicore processor, FPGA, Multimedia processing, High-performance-computing*

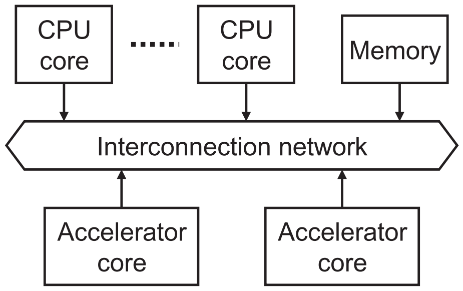


Fig. 1 Heterogeneous multi-core processor architecture.

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, optimal architecture is different from application to application. Heterogeneous multicore processing is proposed to execute applications power-efficiently. It uses different processor cores such as CPU cores and accelerator cores as shown in Fig. 1. If the tasks of an application are correctly allocated to the most suitable processor cores, all the cores work together to increase the overall performances.

Examples of low-power heterogeneous multi-core processors are [1] and [2]. The former has multiple cores of CPUs and ALU arrays. The latter has multiple cores of CPUs, a micro-controller and SIMD (single-instruction multiple-data) type processors. An example of a heterogeneous high-performance computing is “Tianhe-1A” [3] which has Intel X5670 CPUs and NVDIA GPUs. Commercially available heterogeneous multicore processors are partially programmable so that a part of the data path and computations of processing elements (PEs) can be changed to some extent. However, due to the wide variety of tasks and their different memory requirements, the programmability in commercially available processors is not enough to extract sufficient performance. Moreover, the programming environments in various heterogeneous architectures such as embedded processors [2] and CPU/GPU high performance computing [3] are different. Therefore, each time the architecture changes, large design time is required to re-map the application into the new architecture.

To solve these problems, we propose an FPGA-based platform for heterogeneous multicore processors to explore accelerator architectures suitable for applications. 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. The proposed platform consists of CPU cores suitable for control-intensive tasks and custom accelerator cores suitable for data-intensive tasks. For custom accelerator architectures, we employ some typical accelerator architectures used in the previous work [1]–[3] as architecture templates. We consider two types of custom accelerators: SIMD one-dimensional PE array (SIMD-1D) and MIMD two-dimensional PE array (MIMD-2D). The SIMD-1D accelerator is suitable for executing simple operations at a high degree of parallelism. The proposed SIMD-1D accelerator is designed similar to the GPU data path to use the CUDA (compute unified devise architecture) [4] programming language. The MIMD-2D accelerator is suitable for executing complex operation at a medium degree of parallelism. The proposed MIMD-2D architecture is designed similar to the FE-GA (Flexible Engine/Generic ALU Array) [1] to execute multimedia applications power-efficiently. To increase the memory access speed, we introduce a custom hardware called address generation unit (AGU).

The high reconfigurability of FPGAs enables to adopt the different types of accelerators for a single application depending on the nature of tasks. The major disadvantage of FPGA-based heterogeneous processors over the com-

Manuscript received March 15, 2013.
Manuscript revised June 28, 2013.

[†]The authors are with the Graduate School of Information Sciences, Tohoku University, Sendai-shi, 985-8579 Japan.
a) E-mail: takei@ecei.tohoku.ac.jp
b) E-mail: hariyama@ecei.tohoku.ac.jp

DOI: 10.1587/transfun.E96.A.2576

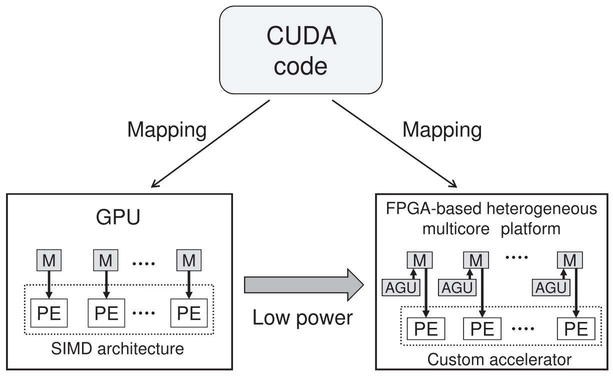


Fig. 2 Mapping of the CUDA code onto GPU and FPGA.

mercially available heterogeneous processors is the low-performance of CPU cores since CPU cores are generated using look-up tables. Such soft-core CPUs cause large computation time and large data transfer time. However, recent FPGAs such as Xilinx Zynq and Altera Cyclone V contain hard-core CPUs operating at about 8 times faster than the soft-core CPUs. This paper is an extension of the work done in [5] which explains the basic idea of the heterogeneous multicore platform. However, the soft-core CPU in [5] is replaced by a low-power hard-core CPU ("Cortex-A9 dual core ARM processor") using Xilinx Zynq, and the times for processing and data transfers are significantly improved compared to the previous works.

The contributions of the proposed FPGA-based heterogeneous multicore platform are summarized as follows. The use of the architecture templates increases productivity in terms of both of hardware and software design. From the point of hardware design, the use of the architecture templates reduces design effort to explore the good architectures suitable for applications. For example, by using the architecture templates, the total processing time can be easily modeled and the design parameters such as the number of accelerator cores can be determined for the given design constraints as described in Sect. 2. From the point of software design, the use of architecture templates would also make it easy to re-use the softwares which are originally made for the accelerators with similar architecture to the templates. For example, our GPU-like SIMD-1D templates allows us to map the CUDA codes easily onto the FPGA-based SIMD-1D accelerator cores [6] as shown in Fig. 2. The resulting FPGA-based processor would be applied to practical low-power embedded systems since the speed and power of the FPGA-based heterogeneous processor is almost comparable to a custom heterogeneous processor [1] from our experimental results.

2. Heterogeneous Multicore Platform

2.1 Overall Architecture

This section explains the architecture of the heterogeneous multi-core platform. Figure 3 shows the overall architecture of the proposed platform. An external memory such as DDRII SDRAM is connected to the CPU core through the FPGA board. The custom accelerators have different architectures such as SIMD-1D and MIMD-2D.

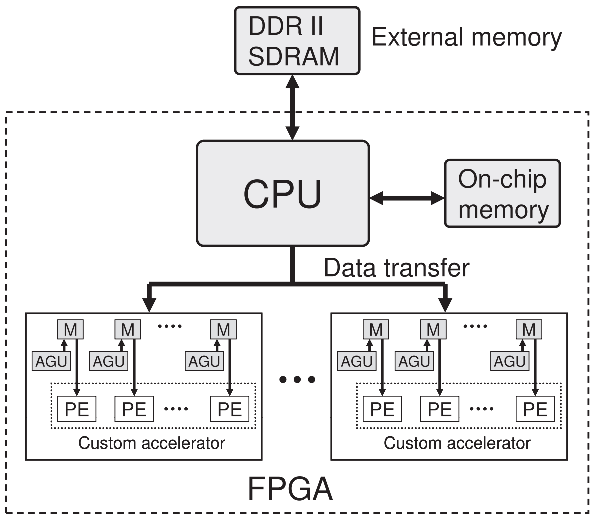


Fig. 3 Proposed heterogeneous multi-core architecture.

Let us consider the latency for memory access. There are two types of memory access on the proposed heterogeneous platform. One is a memory access between PEs and memory modules on the custom accelerators. The other is a memory access between an external memory and memory modules on the custom accelerators. The latency of the former type of memory access is small since both of PEs and memory modules exist on the FPGA. The latency is one cycle on the implemented architecture described in Sect. 4. The latency of the latter type of memory access is large due to the external memory access controlled by the CPU core. The latency is more than ten cycles. It is important to reduce the data-transfer time between CPUs and accelerator cores for a heterogeneous multicore. In previous work [7], the window-based image processing time and a memory capacity are reduced with the optimal memory allocation and the temporally data-transfer scheme.

For further reduction of the total processing time, the data-transfer is overlapped with the computation when two and more accelerators are used as shown in Fig. 4. In order to hide as many data-transfers as possible, the number of accelerator cores N_C is determined by

$$N_C = \left\lfloor \frac{t_{comp}}{t_{trans}} \right\rfloor + 1 \tag{1}$$

where t_{trans} is the data-transfer time and t_{comp} is the computation time of each accelerator core. Figure 5 shows an example of the overlap between the computation on one core and data-transfers of the other cores when $t_{comp} = 3 \times t_{trans}$. In this case, the number of accelerator cores becomes four from Eq. (1).

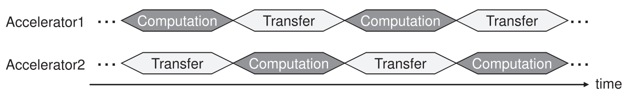


Fig. 4 The overlap of data-transfer and computation.

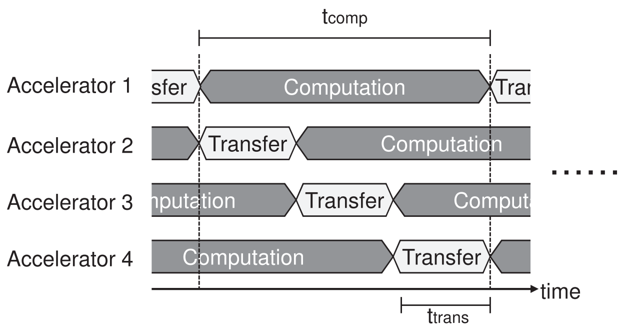


Fig. 5 The overlap between the computation on Accelerator 1 and data-transfers of Accelerator 2, 3 and 4 when $t_{comp} = 3 \times t_{trans}$.

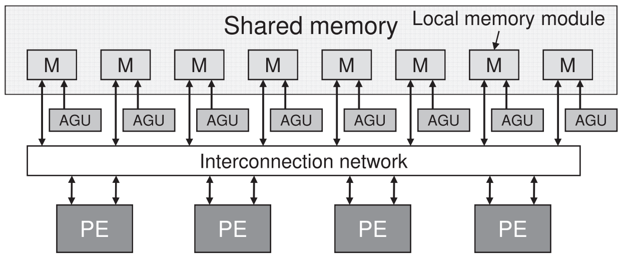


Fig. 6 SIMD-1D architecture.

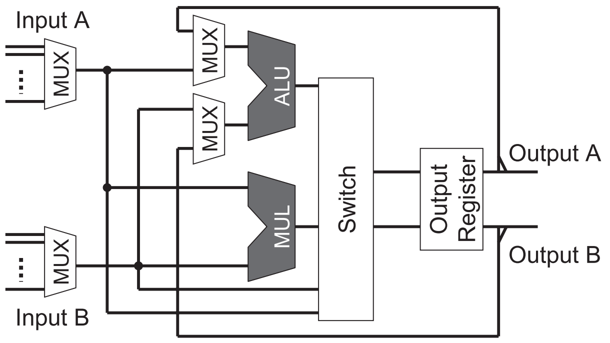
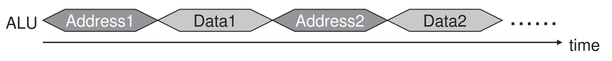


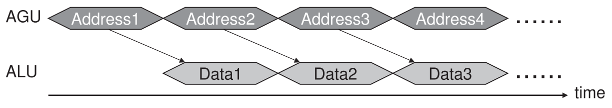
Fig. 7 Architecture of the PE.

Table 1 Operations of a PE.

Operation	Latency (clock cycles)
Addition	1
Subtract	1
Multiplication	1
Accumulation	2
Multiply-accumulation	2
Comparisons	1
Absolute difference	1



(a) Address processing on ALU



(b) Address processing on AGU

Fig. 8 Address processing.

2.2 SIMD-1D Accelerator

The proposed SIMD-1D accelerator is designed similar to the GPU accelerator so that we can use the same CUDA code. The basic idea of the SIMD-1D accelerator is discussed in [6]. It has a 1-dimensional array of PEs connected to the shared memory as shown in Fig. 6. AGUs are included to increase the address generation speed. To execute an application, we have to divide it into independent threads where several of them can be executed in parallel. After the execution is finished, new threads are fed. When all the threads are executed, the resulting data are read by the CPU.

Figure 7 shows the architecture of a PE. It consists of a 16-bit fixed-point ALU and a multiplier. Table 1 shows the operations of a PE. Operations such as addition, accumulation subtraction, comparison and absolute difference computation are done in the ALU, and multiplication is done in the multiplier. Multiply-accumulation is done by a pipelining the multiplier and the adder.

In CPUs, the address calculation and data processing are done in the same ALU as shown in Fig. 8(a). Therefore, when the addresses are calculated, we cannot do data processing. In the proposed architecture, the address calculation is done in the AGU shown in Fig. 8(b). The address calculation and data processing are done in parallel so that we can reduce the total processing time. A detailed description about AGUs is given in [7]. As shown in Fig. 3, accelerators in the proposed heterogeneous platform contain AGUs.

2.3 MIMD-2D Accelerator

The proposed MIMD-2D accelerator is designed based on the FE-GA accelerator [1] that has a dynamically reconfigurable PE array. Figure 9 shows the proposed MIMD-2D accelerator. It consists of a 2-dimensional array of PEs, local memory modules and AGUs. In order to simplify the interconnection network while still meeting the streaming applications, we limit 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. To implement applications, we have to divide it into multiple contexts that execute sequentially. Within a context, we can perform parallel computations. The computation starts after the configuration data of multiple contexts are written to the configuration memory of the accelerator. When the computation is

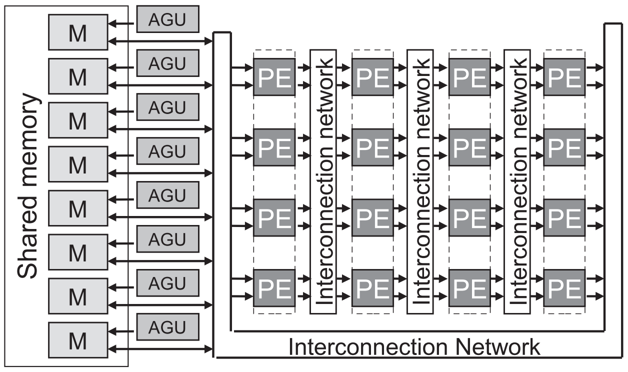


Fig. 9 MIMD-2D architecture model.

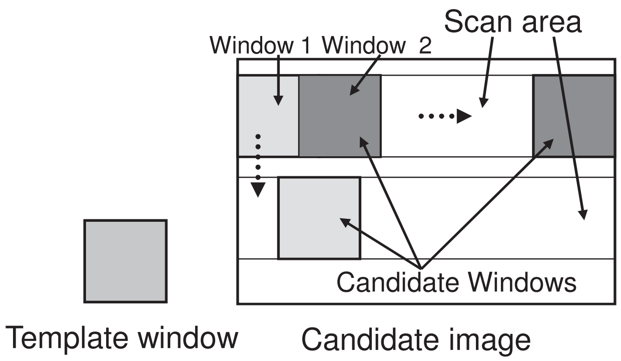


Fig. 10 Template matching.

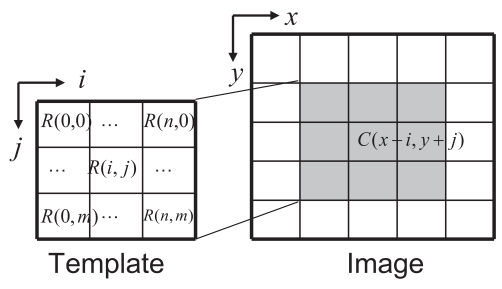
finished, the resulting data are read by the CPU.

3. Mapping Examples

In this section, we discuss the requirements of the applications and their scheduling and mapping methods for different accelerators. We use two mapping examples; SAD-based template matching [8] and filter computation to evaluate the proposed heterogeneous multicore platform. In template matching, a candidate image is scanned to find a segment which is similar to the template. The scanning of the image is done by left to right in scan areas as shown in Fig. 10. The segment of an image is called a "window". To find the most similar window, "sum of absolute difference (SAD)" between the template pixels and the candidate window pixels is calculated as shown in Fig. 11. If a candidate window is similar to the template, the SAD value becomes small. Therefore, we chose the candidate window that gives the smallest SAD as the matching window to the template. The "data flow graph (DFG)" of the template matching is shown in Fig. 12. As shown in Fig. 12, template matching requires two instructions; "absolute difference (AD)" and "addition or accumulation".

In the filter computation, a filter is applied to all candidate windows in an image to produce a new image. The scanning of the image is similar to the scanning in the template matching example shown in Fig. 10. The filter computation is shown in Fig. 13. The DFG of the template matching is shown in Fig. 14. As shown in Fig. 14, filter computation requires a single "multiply-accumulation (MAC)" instruction.

Figures 12 and 14 also show that pixel-level and window-level parallelisms exist for both applications. Moreover, the computations among different windows are independent of each other. Even though two windows are overlapped each other, there are no common results between computations for two windows. Namely, the computations for these windows are completely different from each other. Figure 15 shows an example of two overlapped candidate windows in template matching. The SAD computations of window Cw1 and window Cw2 are given in Eqs. (2) and (3) respectively. In these two SAD computations, there is



$$SAD(x,y)=\sum_{j=0}^{m}\sum_{i=0}^{n}\left|C(x+i,y+j)-R(i,j)\right|$$

Fig. 11 SAD computation.

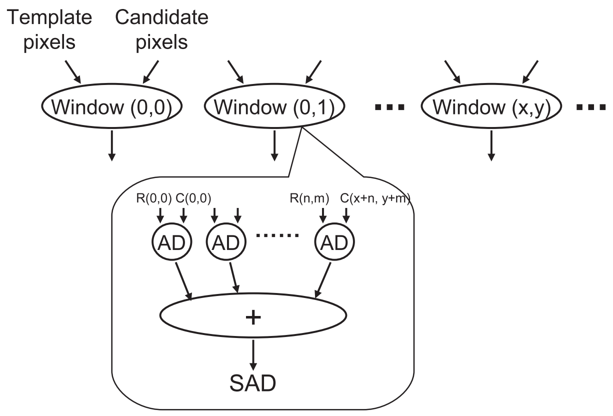
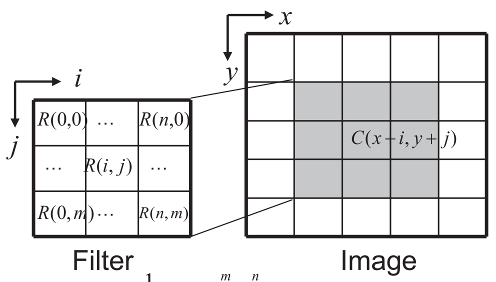


Fig. 12 DFG of the template matching.



$$C(x,y)=\frac{1}{\text{constant}}\times\sum_{j=0}^{m}\sum_{i=0}^{n}C(x+i,y+j)\times R(i,j)$$

Fig. 13 Filter computation.

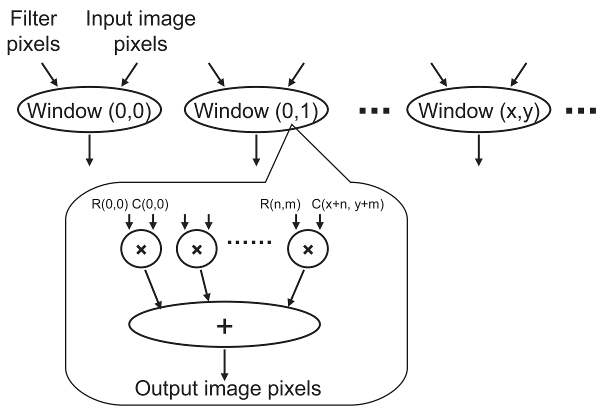


Fig. 14 DFG of the filter computation.

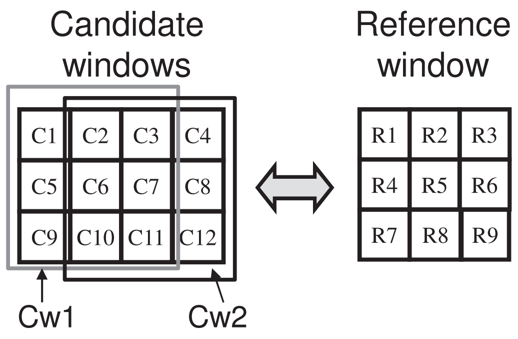


Fig. 15 SAD computation of overlapped windows.

no common absolute difference with the same combination of the reference window data and the candidate window data. Therefore, the computation amount does not increase by computing different windows independently. Similarly, there are no redundant computations in the filter computation.

$$SAD_{Cw1} = |R1 - C1| + |R2 - C2| + |R3 - C3| + |R4 - C4| + |R5 - C5| + |R6 - C6| + |R7 - C7| + |R8 - C8| + |R9 - C9| \quad (2)$$

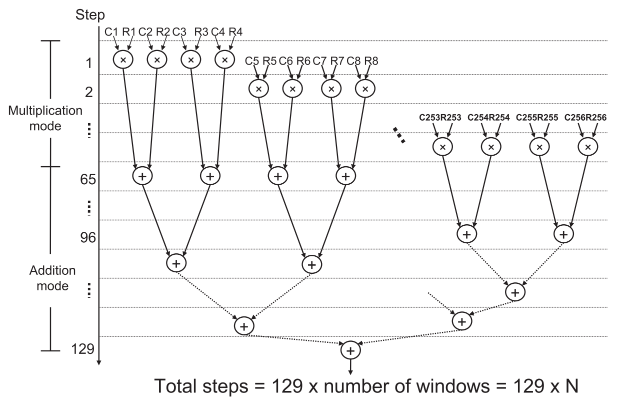
$$SAD_{Cw2} = |R1 - C2| + |R2 - C3| + |R3 - C4| + |R4 - C6| + |R5 - C7| + |R6 - C8| + |R7 - C10| + |R8 - C11| + |R9 - C12| \quad (3)$$

The computations within a window, there is a data dependency. The type of instructions, the degree of parallelism and the data dependency are some of the requirements of the applications. To archive optimal performance, the properties of the accelerators should be optimized for the requirements of the applications under the given design constraints.

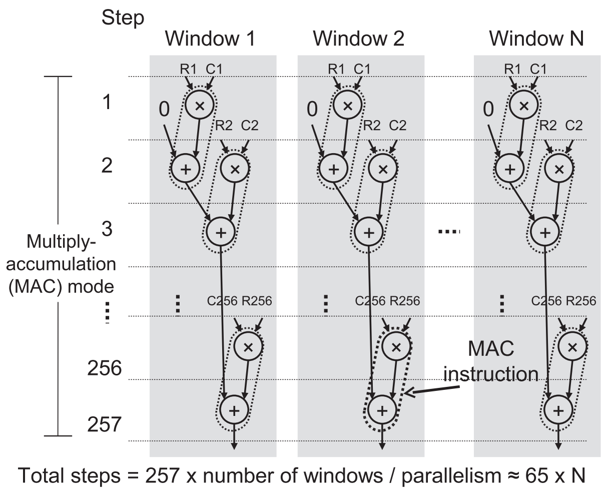
3.1 Mapping for SIMD-1D Architecture

To map an application, we have to consider what kind of parallelism we use. As shown in Figs. 12 and 14, there are two kinds of parallelisms; window-level parallelism and pixel-level parallelism. Figure 16 shows the scheduling of the filter computation for the SIMD-1D architecture when the number of windows is N. Figure 16(a) shows the pixel-parallel scheduling when the degree of parallelism is four. In this scheduling, two instructions; multiplication and addition are used. Figure 16(b) shows the window-parallel scheduling when the degree of parallelism is four. In this scheduling, single MAC instruction is used. Although MAC instruction takes two clock cycles as shown in Table 1, the multiplication and accumulation of different data are pipelined. Multiplication and accumulation instructions are done in parallel as shown in Fig. 16(a). Therefore, window-parallel scheduling is faster than the pixel-parallel scheduling for the filter application. To implement the filter computation in the SIMD-1D architecture, we use window-parallel scheduling.

Figure 17 shows the mapping of the filter computation to the SIMD-1D architecture. The computations among different windows are independent from each other as shown in Fig. 16(b). Computation of one window is performed on each PE. This computation requires the image data and the coefficient data simultaneously as shown in Fig. 16(b). Therefore, we use two memory modules as shown in Fig. 19. One memory module is used to store the coefficient data,



(a) Pixel-parallel scheduling


Step
Window 1
Window 2
Window N
R1 C1
R2 C2
0
C256 R256
Multiply-accumulation (MAC) mode
MAC instruction
1
2
3
256
257
Total steps = 257 x number of windows / parallelism ≈ 65 x N

(b) Window-parallel scheduling

Fig. 16 Scheduling of the filter computation for SIMD-1D architecture (The degree of parallelism=4, The number of windows=N).

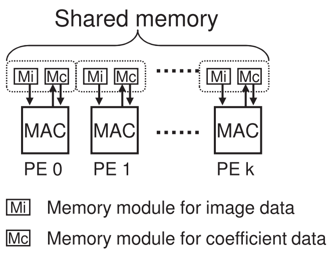
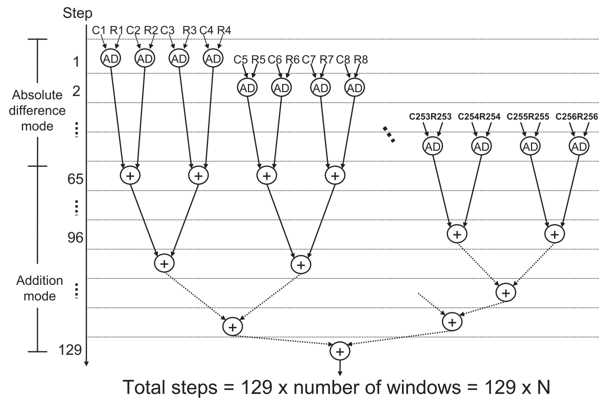
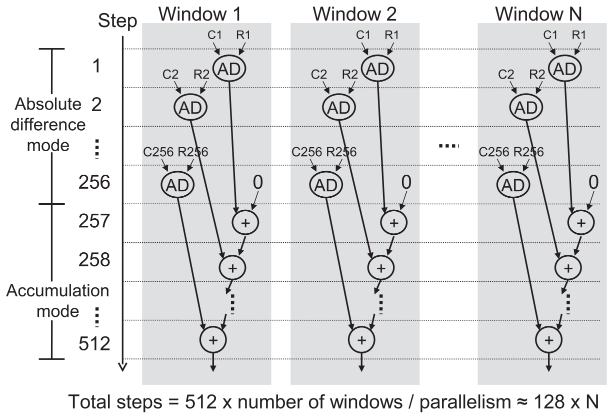

Shared memory
Mi
Mc
MAC
PE 0
PE 1
PE k
Memory module for image data
Memory module for coefficient data

Fig. 17 Mapping of the filter computation on SIMD-1D.



(a) Pixel-parallel scheduling



(b) Window-parallel scheduling

Fig. 18 Scheduling of the SAD computation for SIMD-1D architecture (The degree of parallelism=4, The number of windows=N).

while other memory module is used to store the image data. The resulting data is written to the unused space of the memory module for the coefficient data. This gives a simple interconnection network between the memory and the PE array.

Figure 18 shows the scheduling of the SAD computation for the SIMD-1D architecture when the number of windows is N. Figure 18(a) shows the pixel-parallel scheduling when the degree of parallelism is four. In this scheduling, two instructions, AD and addition, are used. Figure 18(b) shows the window-parallel scheduling when the degree of parallelism is four. In this scheduling, two instructions, AD and addition, are also used. Both scheduling schemes require similar number of control steps. In the pixel-parallel scheduling, we have to distribute the AD computation results among different PEs. Therefore, each PE accesses multiple memory modules to retrieve AD computation results. This gives a complex interconnection network between the memory and the PE array. In the window-parallel scheduling, the computations among different windows are independent of each other and each PE accesses only two memory modules. This gives a simple interconnection network between the memory and the PEs array. Therefore, we use window-parallel scheduling for the implementation of the SAD computation in SIMD-1D architecture,

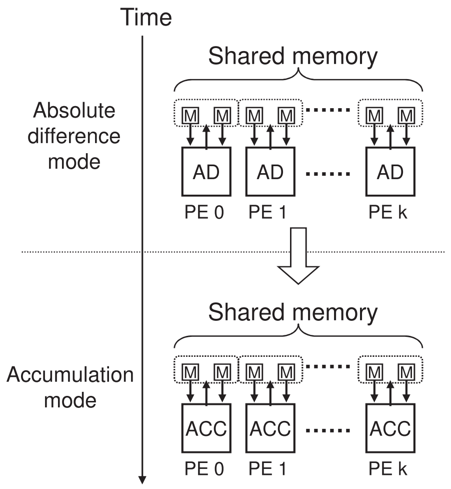


Fig. 19 Mapping of the SAD computation on SIMD-1D.

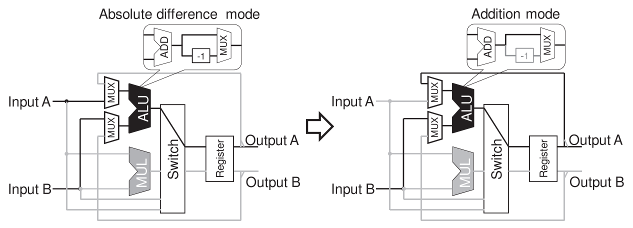


Fig. 20 Dynamic reconfiguration of the PE.

Figure 19 shows the mapping of the filter computation to the SIMD-1D architecture. Since we use two instructions, SIMD-1D architecture executes the AD instruction first and the resulting data are written to the memory. Then, it executes the accumulation instruction for the AD computation results. Since the computations among different windows are independent from each other as shown in Fig. 18(b), each PE reads and writes to the same memory modules.

To implement the processing on the SIMD-1D architecture, the dynamic reconfiguration is used in PEs and AGUs. In PEs, the operation of the ALU and the data path are changed as shown in Fig. 20 when the SAD computation is performed. The used data path and the unused data path are denoted black and gray, respectively. In AGUs, the base address is changed in order to access the data in different windows. The dynamic reconfiguration of AGUs is

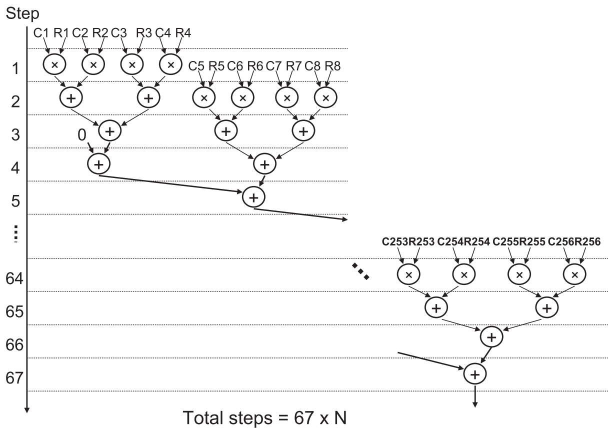


Fig. 21 Pixel-parallel scheduling of the filter computation for MIMD-2D architecture (The degree of parallelism=4, The number of windows=N).

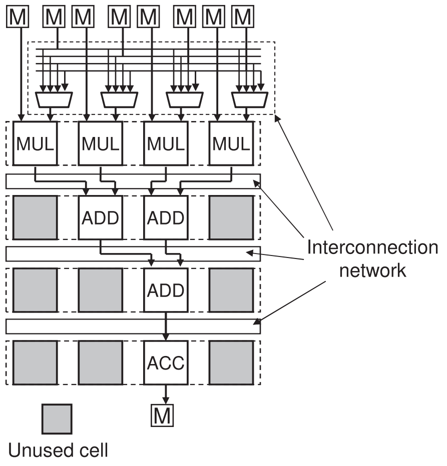
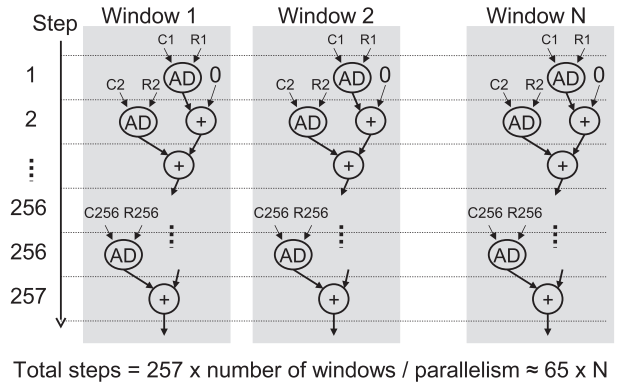
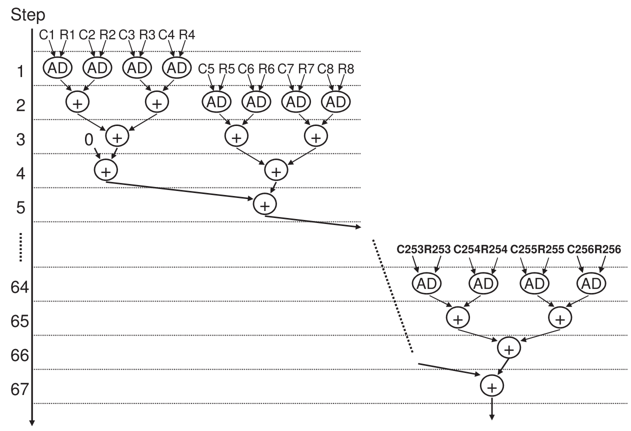


Fig. 22 Mapping of the filter computation on MIMD-2D.


Step
Window 1
Window 2
Window N
Total steps = 257 x number of windows / parallelism ≈ 65 x N

(a) Window-parallel scheduling



(b) Pixel-parallel scheduling

Fig. 23 Scheduling of the SAD computation for MIMD-2D architecture (The degree of parallelism=4, The number of windows=N).

explained in detail in [7].

3.2 Mapping for MIMD-2D Architecture

Figure 21 shows the pixel-parallel scheduling of the filter computation for MIMD-2D architecture, when the degree of parallelism is four and the number of windows is N. In this scheduling, multiple instructions, multiplications and additions, are used in different PEs at the same time. The window-parallel scheduling for MIMD-2D architecture is the same as that for the SIMD-1D architecture shown in Fig. 16(b). Both of these scheduling schemes take almost the same number of control steps (or clock cycles) to execute. Therefore, we can use either of the scheduling schemes.

For the simplicity, we explain the mapping using the pixel-parallel scheduling. Figure 22 shows the mapping result on the MIMD-2D accelerator. Multiplication, addition and accumulation instructions are assigned to 8 PEs. The rest of the PEs is unused. Since the parallelism is limited by the number of memory modules, we cannot use those PEs to increase the processing speed.

Figure 23 shows the scheduling of the SAD computation for the MIMD-2D architecture when the number of windows is N. Figures 23(a) and 23(b) show the window-parallel and pixel-parallel scheduling schemes of the SAD computation for MIMD-2D architecture, when the degree of parallelism is four, both of these scheduling schemes take almost the same number of control steps to execute. Therefore, we can use either of the scheduling schemes.

For the simplicity, we explain the mapping using the pixel-parallel scheduling. As shown in Fig. 24, 50% of the PEs are used for AD computation, addition and accumulation instructions. The rest of the PEs are unused. Since the parallelism is limited by the number of memory modules, we cannot use those PEs to increase the processing speed.

To implement the processing on the MIMD-2D architecture, the dynamic reconfiguration is used in AGUs and the interconnection network between PEs and memory modules. In AGUs, the base address is changed in order to access the data in different windows. The interconnections between PEs and memory modules are changed as shown in Fig. 25 when the scan area shown in Fig. 10 changes. The dynamic reconfiguration of the interconnection is explained in detail in [7].

In these mapping examples, we can say that the num-

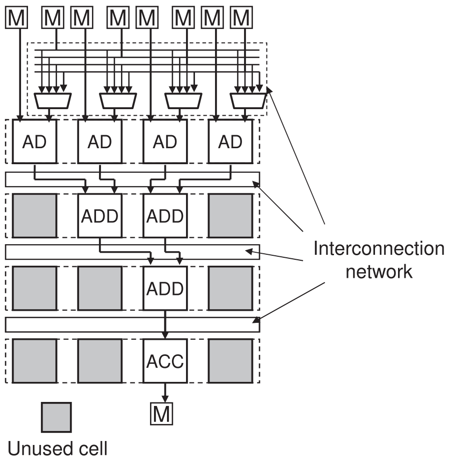


Fig. 24 Mapping of the SAD computation on MIMD-2D.

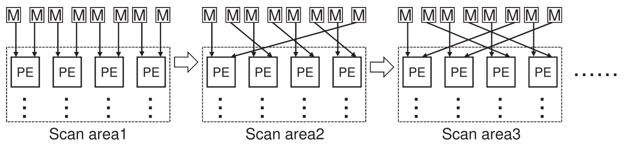


Fig. 25 Changing the interconnection of MIMD-2D.

Table 2 Specification of accelerator cores.

Accelerator core	Number of PEs	Number of LUTs	Number of memories	Degree of parallelism
SIMD4	4 × 1	3301	8 (16kB)	4
SIMD9	9 × 1	7354	18 (18kB)	9
MIMD12	4 × 3	7322	8 (16kB)	4

Table 3 Comparison 1 : The same number of LUTs.

Application	Accelerator core	Processing time (ms)
Filter	SIMD9	0.069
	MIMD12	0.154
SAD	SIMD9	0.139
	MIMD12	0.154

Table 4 Comparison 2 : The same number of memory modules.

Application	Accelerator core	Processing time (ms)
Filter	SIMD4	0.156
	MIMD12	0.154
SAD	SIMD4	0.318
	MIMD12	0.154

ber of parallel data accesses, number of PEs, the PE array structure and the functions of the PEs are some of the main properties of the accelerators. In the next section, we evaluate the performance of the two accelerators to identify some relationship with the application requirements.

4. Evaluation

We implement the proposed heterogeneous multicore platform on Xilinx Zynq-7000 EPP ZC702 evaluation kit [9]. The SIMD-1D and MIMD-2D architectures have different topologies and different number of PEs. Therefore, we do two comparisons to evaluate the architectures. In the first comparison, the number of look-up-tables (LUTs) in both accelerators is a constant. We do not use any DSP units in this evaluation to compare the accurate LUT usage. In the second comparison, the degree of parallelism of the memory access is a constant. In parallel processing, both the number of PEs and the degree of parallelism with the memory are equally important. We implement three types accelerator cores shown in Table 2 on the FPGA. As shown in Table 2, SIMD9 and MIMD12 accelerators have almost the same number of LUTs. SIMD4 and MIMD12 accelerators have the same number of memory modules. Therefore, the degree of parallelism of the memory access is the same.

We compare the processing time of filter computation and SAD computation on accelerator cores. For test, the image size is 256×16. The window size is 16×16. The frequency is 100 MHz. Block memories on the FPGA are used for local memory modules of the accelerator cores. The latency of memory access is one cycle.

Table 3 shows the comparison of SIMD-1D (SIMD9) and MIMD-2D (MIMD12) accelerators when the number of LUTs is a constant. For the filter computation, the processing time of the SIMD-1D accelerator is less than half of that of the MIMD-2D accelerator. The SIMD-1D accelerator has a one-dimensional PE array, where all 9 PEs are directly connected to the memory as shown in Fig. 9. The MIMD-2D architecture has a two-dimensional PE array with 4×3 where only leftmost 4 PEs can directly retrieve from the local memory modules as described in Sect. 2.3. Therefore, the SIMD-1D accelerator has the higher degree of parallelism of memory access than the MIMD-2D accelerator. As a result, the SIMD-1D accelerator is more suitable for the filter computation than the MIMD-2D accelerator. In the SAD computation, the SIMD-1D accelerator is slightly faster than the MIMD-2D accelerator. As explained in Sect. 3, SAD computation requires two types of operations: absolute difference and addition. The SIMD-1D accelerator cannot perform these two types of operations at a same time, while the MIMD-2D accelerator can perform these operations at a same time by pipelining. Regardless of pipelining in the MIMD-2D accelerator, the processing time of the SIMD-1D accelerator is still smaller due to its high degree of parallelism of memory access.

Table 4 shows the comparison of SIMD-1D (SIMD4) and MIMD-2D (MIMD12) accelerators when the degree of parallelism of the memory access is constant; the number of memory modules is four. In the filter computation, the processing times of the SIMD-1D and MIMD-2D accelerators are the same. Multiplications and additions can be well pipelined on both of SIMD-1D and MIMD-2D accelerators as shown in Figs. 16 and 21. Hence, the processing time entirely depends on the degree of parallelism of memory access. Since both accelerators have the same degree of parallelism of memory access, the processing time is the same. In the SAD computation, the processing times of MIMD-2D accelerator is about half of the SIMD-1D accelerator. As described above, the MIMD-2D accelerator can pipeline different type of operations (absolute difference and addition in SAD computation). Hence, MIMD-2D can obtain a higher degree of parallelism of operations under the condition of the number of memory modules than the SIMD-1D accelerator. If we have an application with more types of operations, the MIMD-2D accelerator could give much better results than the SIMD-1D accelerator.

As explained in Sect. 3, applications have various requirements such as the data dependency, the degree of parallelism, different types of instructions etc. The filter computation example has only one instruction and requires a lot of parallel data access. In such cases, SIMD-1D architecture is suitable, since it has a very small control overhead and all PEs are connected to the memory for parallel data access. The SAD computation example has two operations and also require a lot of parallel data access. In this particular example, when the degree of parallelism is large, SIMD-1D gives the better results. When the degree of parallelism is the same, MIMD-2D gives the better results. Therefore, if the number of instructions in an application is large, MIMD-2D architecture become more suitable.

To select the best accelerator for a given application, we have to match the requirements of the application with the properties of the accelerator under the design constraints. As explained in Sects. 3 and 4, most of the application requirements and accelerator properties can be parameterized and represented mathematically. The design constraints are the operating frequency, amount of hardware resources such as LUTs and memories, power consumption, etc. Our next step would be to find a mathematical relationship between those application requirements and the accelerator properties to satisfy the design constraints. Then we can automatically optimize the proposed heterogeneous platform for given applications.

Let us compare the FPGA-based platform with a conventional custom heterogeneous multicore processor. We implement the proposed architecture on Xilinx Zynq-7000 EPP ZC702 evaluation kit as explained above. Zynq integrates a dual-core Cortex-A9 CPU and FPGA equivalent to Atrix-7 on a single chip. In addition, the evaluation kit has a DDR3 SDRAM for an external memory.

Figure 26 shows the implemented architecture on the evaluation kit. There are MIMD-2D accelerator cores which process filter computation in parallel, and the control unit which deal with start/stop signal. Table 5 shows the resource utilization on the FPGA with four MIMD16 cores. DSP48 in Table 5 is a 48-bit DSP unit in a Xilinx FPGA. It is used for a multiplier unit in the PE shown in Fig. 7. Since the FPGA design tool removes unused units on the implemented architecture automatically, the resource utilization is smaller than expected. Note that the number of accelerator cores and the number of PEs in one core can be selected depending on applications. As shown in Fig. 27, the width of AXI interconnection on Zynq is 32 bit. Therefore, we can transfer four 8-bit pixels simultaneously and reduce the total data-transfer time.

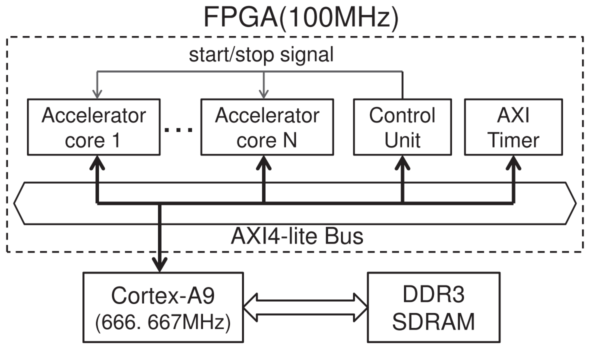


Fig. 26 Implemented architecture.

Table 5 Resource utilization with 4 MIMD16 cores.

Module	LUT	Register	Block memory	DSP48*
Accerelators	1044	1604	18	16
Controll unit	28	28	0	0
AXI timer	312	217	0	0
AXI Interconnect	397	182	0	0
Total	1781(3%)	2031(2%)	18(13%)	16(7%)

*DSP48 is a 48-bit DSP unit in a Xilinx FPGA.

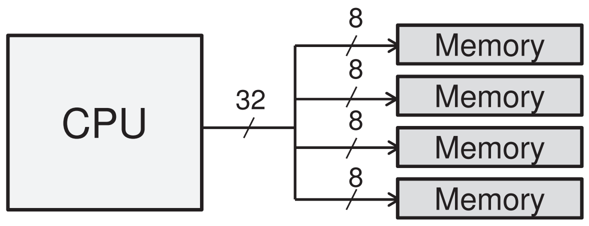


Fig. 27 Parallel transfer on the AXI4 bus.

Table 6 shows the comparison of the filter computation time of the proposed FPGA-based platform and RP1 [1]. The image size is 640×480. The window size is 12×12, 18×18 and 24×24. The number of PEs on the FPGA-based platform is 64, and it is equal to that of two FE-GA cores. We estimate the power consumption of the FPGA-based platform by using Xilinx Power Estimator 14.3. Comparing the processing time, when the number of FE-GA cores is one, the processing time of the proposed platform is smaller than that of RP1 in any window size. Moreover, when the window size is 18×18, the processing time of the proposed platform is smaller than that of RP1 with two FE-GA cores. The power consumption of the proposed platform is smaller than that of RP1 as shown in Table 7. In conclusion, the FPGA-based heterogeneous multicore architecture is comparable to custom heterogeneous multicore processors. Moreover, we can reduce the processing time by using more accelerator cores.

5. Conclusion

We have proposed an FPGA-based heterogeneous multicore

Table 6 Comparison of speed with RP1.

Window size	Processing time on Zynq(ms)	Processing time on RP1(ms) [7]	
	1xCortex-A9(666.667 MHz) +FPGA(100 MHz)	1xSH-4A(600 MHz) +1xFE-GA(300 MHz)	1xSH-4A(600 MHz) +2xFE-GA(300 MHz)
12×12	46.51	47.72	36.24
18×18	70.50	102.67	72.94
24×24	115.89	137.07	96.55

Table 7 Comparison of power consumption with RP1.

	Zynq	RP1 [7]	
	1xCortex-A9(666.667 MHz) +FPGA(100 MHz)	1xSH-4A(600 MHz) +1xFE-GA(300 MHz)	1xSH-4A(600 MHz) +2xFE-GA(300 MHz)
Power consumption	1.03 W	1.30 W	1.36 W
Process technology	28 nm [10]	90 nm [1]	

platform with custom accelerators. The accelerator cores are customizable for each application. Dedicated AGUs are used to increase the processing speed and to reduce the area and power. In this paper, we evaluate the proposed platform using several examples. We discussed many key requirements of the applications and the properties of the accelerators. We also discussed several scheduling schemes and mapping methods and analyze their performance. Such an evaluation is very important to find a relationship between the requirements of the applications and properties of the accelerators. Since the heterogeneous computing have a wide variety of applications from low-power processing to high-performance-computation, finding this relationship is very essential to optimize their performance.

Acknowledgment

This work is supported by MEXT KAKENHI Grant Number 12020735.

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 multi-core 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] http://www.top500.org/system/10587

[4] NVIDIA Corporation, “NVIDIA CUDA programming guide,” Ver2.2.1, 2009.

[5] H.M. Waidyasooriya, Y. Takei, M. Hariyama, and M. Kameyama, “FPGA implementation of heterogeneous multicore platform with SIMD/MIMD custom accelerators,” IEEE International Symposium on Circuits and Systems (ISCAS), pp.1339–1342, 2012.

[6] H.M. Waidyasooriya, M. Hariyama, and M. Kameyama, “Architecture of an FPGA-oriented heterogeneous multi-core processor with SIMD-accelerator cores,” International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), pp.179–186, 2010.

[7] 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.

[8] 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, July 2005.

[9] http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm

[10] S. Dutta, V. Rajagopolan, B. Taylor, and R. Wittig, “Xilinx Zynq embedded processing platform,” A Symposium on High Performance Chips (HOT Chips 23), 2011.

Yasuhiro Takei received the B.E. degree in electronic engineering, 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 a postdoctoral researcher in Graduate School of Information Sciences, Tohoku University. His research interests include heterogeneous multicore processor architectures and high-level design methodology for VLSIs.

Masanori Hariyama received the B.E. degree in electronic engineering, M.S. degree in Information Sciences, and Ph.D. in Information Sciences from Tohoku University, Sendai, Japan, in 1992, 1994, and 1997, respectively. He is currently an associate professor in Graduate School of Information Sciences, Tohoku University. His research interests include VLSI computing for real-world application such as robots, 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 Dean and 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.