

# Platform and Mapping Methodology for Heterogeneous Multicore Processors

Masanori HARIYAMA\*, Hasitha Muthumala WAIDYASOORIYA,  
Yasuhiro TAKEI, and Michitaka KAMEYAMA

*Graduate School of Information Sciences, Tohoku University,  
6-6-05, Aoba, Aramaki, Aoba, Sendai 980-8579, Japan*

Received May 31, 2012; final version accepted August 29, 2012

Heterogeneous multi-core processors are attracted by various type of applications from low-power media applications to high-performance computing due to their capability of drawing strengths of different cores to improve the overall performance. However, the data transfer bottlenecks between different cores becomes a serious problem. This paper presents two key methodologies to solve the data transfer bottleneck: memory allocation considering an addressing function constraint and task allocation based on algorithm transformation. Moreover, in order to help to explore accelerator architecture suitable for applications, this paper presents a platform based on FPGAs where circuitry is reconfigured by users after fabrication.

**KEYWORDS:** heterogeneous multi-core processor, reconfigurable architecture, dynamic reconfiguration, task-allocation, memory allocation

## 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. The optimal architecture is different from application to application. Heterogeneous multicore processing is proposed to execute different 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 NVIDIA GPUs.

When mapping applications onto a heterogeneous processor, the major issue is the data transfer bottleneck. In heterogeneous multi-core processors, different tasks are allocated to different processor cores. Therefore, processor cores need to access each other’s data through the interconnection network. The accelerator cores contain several local memories for parallel data access. Therefore, whenever a data transfer with an accelerator core occurs, we need to transfer data from the global memory to the accelerator core’s local memories. As a result, a large amount of time is wasted for data transfers and that increases the total processing time. To solve data-transfer bottleneck, this paper presents the two key technologies: memory allocation and task allocation to minimize the data-transfer amount. In embedded heterogeneous processors such as [1], the address generation units (AGU) are integrated with the memory

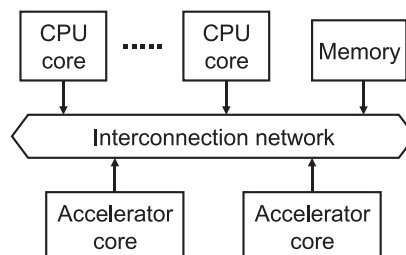


Fig. 1. Heterogeneous multi-core processor architecture.

\* Corresponding author. E-mail: hariyama@ecei.tohoku.ac.jp

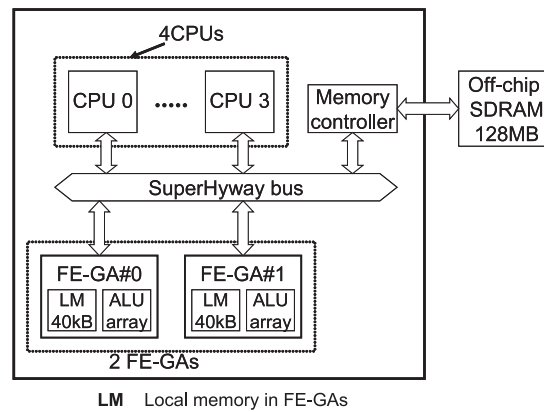


Fig. 2. Block diagram of RP1.

modules. Since the function of the AGU is limited to the simple ones such as stride access, data duplication is caused for repeatedly-accessed data. Therefore, the proposed memory allocation optimizes the parameters of the addressing function together with data location. Moreover, in the task allocation, we consider the algorithm transformation to allocate the successive tasks with data dependencies to the same processor core, so that one task's output is immediately used by the other tasks without transferring it to the global memory.

Another major issue in heterogeneous multicore processors is to explore accelerator architecture suitable for applications. 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 heterogeneous multicore platform. Recent FPGAs have over 300,000 logic cells, over 2 M Bits of memory and high-speed data transfer interfaces. Therefore, a single FPGA is large enough to hold hundreds of processor cores. The proposed platform consists with CPU cores suitable for control-intensive tasks and custom accelerator cores suitable for data-intensive tasks. In this paper, we consider two types of custom accelerators; SIMD 1-dimensional PE array (SIMD-1D) and MIMD 2-dimensional PE array (MIMD-2D); SIMD-1D accelerator is suitable for executing simple operations at a high degree of parallelism, and MIMD-2D accelerator for executing complex operation at a medium degree of parallelism.

## 2. Mapping Technology for Heterogeneous Multicore Processors

### 2.1 Architecture model

We use the heterogeneous architecture [1] that has 4 homogeneous CPUs (SH cores) and 2 FE-GAs. Figure 2 shows the overall architecture of the processor. An off-chip SDRAM of 128 MB is attached to the "SuperHyway" bus to store a large amount of image data receiving from image devices such as cameras. The processing speeds of FE-GAs and CPUs are 300 and 600 MHz respectively.

The architecture of the FE-GA is shown in Fig. 3. It has an array of 32 processing elements (PEs) or cells. These cells are dynamically reconfigured as adders, subtractors, logic gates etc. A PE can be connected to its four neighbors and those connections are also dynamically reconfigurable. The FE-GA has multiple banks of local memory called CRAMs (compiled random access memories). Each CRAM has 2 ports and 4 kByte memory capacity. There are 10 CRAM modules. The PEs at the right and left borders of the FE-GA can be connected to the CRAMs through a crossbar network. The LS (Load/Store) cells contain address generation units. The parameters of the addressing function are the "base address," the "increment" and the "number of iterations." At the beginning, the address is set to the *base address*. In each clock cycle, address increased by the amount of *increment*. When the number of cycles equals to the *number of iterations*, the address is reset to the *base address*. This is called a linear addressing function and is not capable of doing complex addressing. It is also very difficult to use the PE array to create complex addresses, since there are only 32 PEs available in the array.

In this architecture, the crucial problem in performance is the low data transfer rate (memory bandwidth) between the off-chip SDRAM and the local memories of the accelerator (the FE-GA). By measuring the effective data transfer rate between them under the random access condition, the data transfer rate is evaluated to be 670 bps (Bits Per Second). This low data transfer rate causes the serious data-transfer bottleneck. Hence, the technologies to reduce data transfers between the off-chip memory and the local memories of the accelerator is one most important issue.

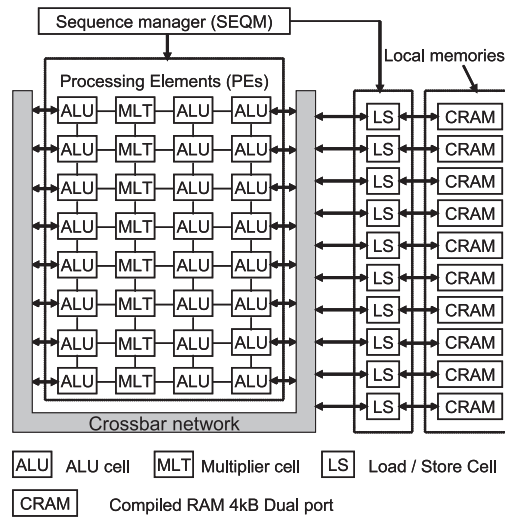


Fig. 3. Structure of the FE-GA.

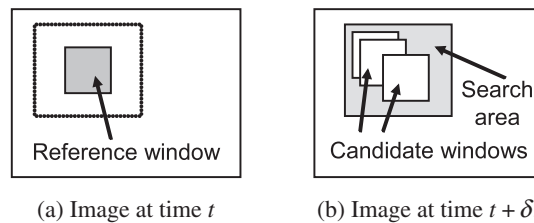


Fig. 4. Corresponding points search in the optical-flow extraction.

## 2.2 Memory allocation for data-transfer minimization considering addressing function's parameters [5]

### 2.2.1 Example: optical-flow extraction

The motions of objects are extremely efficient information to recognize the real-world environment in applications such as robotics and human interfaces. The optical flow of an object in an image refers to the motion vectors of all the pixels of that object. In optical flow extraction, corresponding pixels between two images taken at time  $t$  and  $t + \delta t$  are searched. To find a corresponding pixel, a reference window of a particular pixel on the image at time  $t$  and a search area on the image at time  $t + \delta t$  are considered as shown in Fig. 4. Different candidate windows are selected from the search area and SAD (sum of absolute differences) with the reference window is calculated. The similar the reference window to the candidate window is, the smaller the SAD becomes. Therefore, the candidate window when the SAD becomes minimum is selected as the corresponding window to the reference window.

### 2.2.2 Memory allocation

To solve the data transfer problem, we need to reduce the transferred data amount. As shown in Fig. 4(b), all the candidate windows overlap each other. Therefore, if we allocate the shared area effectively, we can reduce the transferred data amount and also the data transfer time. To explain this problem, we consider an example that has a search area and candidate windows of sizes  $10 \times 10$  and  $4 \times 8$  respectively. We consider there are only four memory modules or CRAMs. Figure 5(a) shows the coordinates of the pixels in the image and Fig. 5(b) shows the memory allocation.

To allocate the data effectively, we consider the access of the candidate windows. As shown in Fig. 5(a), window 1 to 7 are accessed by moving the windows horizontally. In each movement, we access a new vertical line. For example, when window 2 is accessed, we need the data of the vertical line [0,4 to 7,4]. Therefore, we allocate the data of the vertical lines one-by-one to CRAMs as shown in Fig. 5(b). To access the data of the first window, we have to access address 00 to address 07 in 4 CRAMs. Similarly, we access the addresses 02 to 09 for the second window. To access the window 8, candidate window moves one pixel down. The difference between window 1 and window 8 is the horizontal line [7,0 to 7,3]. All the other data are the same for both windows. We have the data of lines 1 to 3 and 5 to 7 in CRAM1 to CRAM3. Therefore, we allocate the data of lines 4 and 8 to the next available address (address 20) of CRAM0.

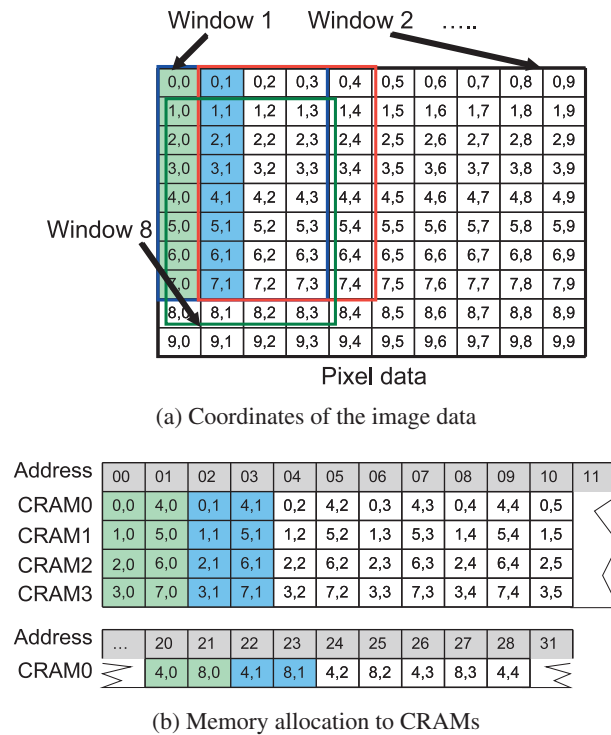


Fig. 5. Memory allocation.

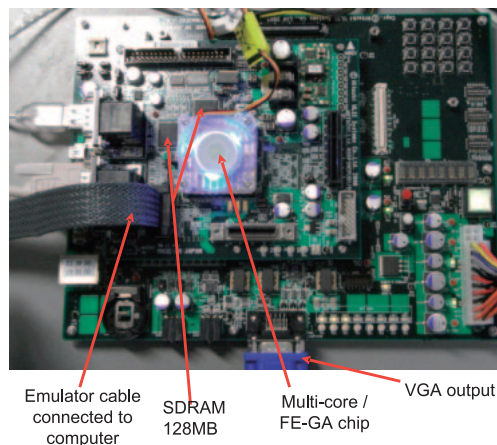


Fig. 6. Heterogeneous multi-core processor board.

### 2.2.3 Evaluation

We used the processor board shown in Fig. 6 for the evaluation. We use a search area and candidate windows of sizes  $24 \times 24$  and  $16 \times 16$  respectively. The image size is  $640 \times 480$ . The gap between two reference windows is 20 pixels. The amount of data transfer is reduced to less than 5% compared to the conventional method using the linear addressing function.

Figure 7 shows the power and speed-up of different implementations using different number of CPUs and FE-GAs. According to the results, 20 times speed-up is achieved using a single FE-GA. The speed-up is 39 times for 2 FE-GAs. If the frame rate is 30 fps, we can achieve a real-time processing using a combination of CPU and FE-GAs.

For further reduction of data-transfers, the proposed memory allocation is extended to exploit not only spatial locality but also temporal locality [7].

## 2.3 Task allocation based on algorithm transformation [8]

### 2.3.1 Overview

In the task allocation method, we make a library that contains CORDIC algorithms [9] for accelerator-incompatible calculations. The task allocation method starts with an initial solution by assigning data-parallel and accelerator-

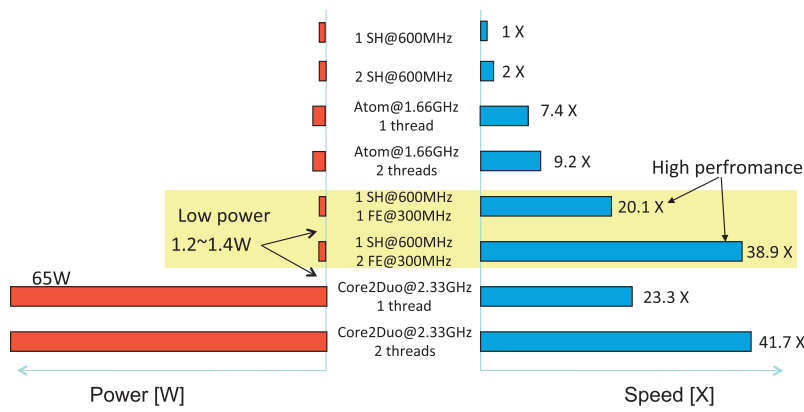


Fig. 7. Power and speed.

compatible tasks to the accelerator cores and the rest to the CPU cores. Then we estimate processing times and the data transfer times for all the tasks. Since the calculation time depends on the amount of data, the number of control steps, the number of parallel calculations and the clock frequency, we can approximate it at high-level-synthesis. Similarly we approximate the data transfer time using the amount of data and the data transfer rate. We create a priority list that contains processing time and data transfer time information in the descending order and select the most critical processing time or the data transfer time from the top of the priority list. If it is a processing time, we consider allocating that task to a different core. If it is a data transfer time, we re-allocating the data dependent tasks either to the CPU core or to the accelerator core. When assigning tasks with accelerator-incompatible calculations, we refer the library and pick a CORDIC algorithm. After the task allocation, we evaluate the data transfer time and the calculation time. If the total processing time decreased, we accept the new assignment and update the priority list. If the total processing time increased, we reject the new assignment and remove the first entry from the priority list. Then we update the priority list and move to the next most critical processing or data transfer time. When the priority list is empty, we terminate the optimization.

### 2.3.2 Example: HOG algorithm

We use HOG (histogram of oriented gradients) algorithm [4] to evaluate the proposed task allocation method. The HOG feature descriptor is widely used in computer vision and image processing for object detection. It contains 4 major tasks.

**Task 1: Gradient computation:** A 1-D derivative mask is applied to all the pixels in the image in horizontal and vertical directions. Equations (1) and (2) gives the horizontal and vertical derivatives respectively. The intensity of the pixel at the coordinates  $x, y$  is given by  $I(x, y)$ .

$$f_x(x, y) = I(x + 1, y) - I(x - 1, y), \quad (1)$$

$$f_y(x, y) = I(x, y + 1) - I(x, y - 1). \quad (2)$$

**Task 2: Orientation binning:** Each pixel within the cell casts a weighted vote given by eq. (4) for an orientation-based histogram channel using the values found in the gradient computation. The orientation is given by eq. (3).

$$m(x, y) = \sqrt{f_x(x + 1, y)^2 + f_y(x - 1, y)^2}, \quad (3)$$

$$\theta(x, y) = \tan^{-1} \frac{f_y(x, y)}{f_x(x, y)}. \quad (4)$$

**Task 3: Descriptor blocks:** In this task, the cells are grouped together to make larger blocks.

**Task 4: Block normalization:** If  $V_k$  is the vector of the block region and  $\epsilon$  is a very small value close to zero, the HOG descriptor  $V$  is given by eq. (5).

$$V = \frac{V_k}{\sqrt{\|V_k\|^2 + \epsilon}}. \quad (5)$$

### 2.3.3 Evaluation

We used a heterogeneous multi-core processor explained in [1]. The accelerator contains 32 processing elements that are capable of additions, multiplications, etc. However, complex operations such as square-root, division, trigonometric calculations are not available. Since tasks 2 and 4 of the HOG algorithm contain accelerator-incompatible square-root and trigonometric operations, we map them to the CPU cores. The tasks 1 and 3 contain data-parallel processing and

Table 1. Power and energy consumption.

	Processing time ( $\mu$ s)	Power (W)	Energy (J)
CPU	5554	1.21	$6.72 \times 10^{-3}$
CPU and accelerator (conventional method)	1626	1.30	$2.11 \times 10^{-3}$
CPU and accelerator (proposed method)	330	1.30	$0.43 \times 10^{-3}$

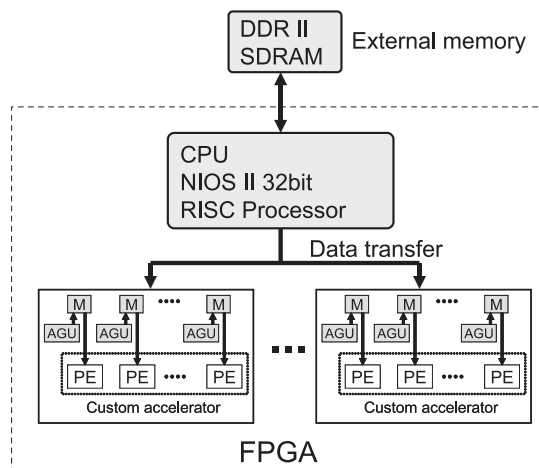


Fig. 8. Proposed heterogeneous multi-core architecture.

are mapped to the accelerator cores. Then we perform the optimization process as explained above. In this particular example, allocating all the tasks to the accelerator core gives the largest processing speed.

Table 1 shows the comparison of processing time, power and energy consumptions. By using the conventional task allocation, we gain 70% reduction of processing time. Using the proposed task allocation method. The power consumption has lightly increased when the accelerator core is used. However, the energy is decreased significantly due to the reduction of the processing time. In the proposed task allocation, the energy consumption reduced by 93% compared to CPU-only implementation.

### 3. FPGA-Based Platform for Heterogeneous Multicore Processors

#### 3.1 Overall architecture [10]

This section explains the architecture of the heterogeneous multicore platform. Figure 8 shows the proposed heterogeneous multi-core platform. It consists of CPU cores, on-chip memory and custom accelerator cores. An external DDR II SDRAM is connected to the CPU cores through the FPGA board. The custom accelerators have different architectures such as SIMD-1D and MIMD-2D.

Since our idea is to allocate different tasks to their most suitable processor core/unit, we allocate the memory address generation to dedicated AGUs. Address generation usually requires complex operations such as multiplications, divisions and modulo operations. Therefore, usually the AGU overhead is large due to the complex hardware units such as dividers and modulo calculators. Such large hardware overhead can be reduced by exploiting the regular memory access patterns in many applications. Examples of such a regular access patterns is the linear access where the address increases linearly. Exploiting these patterns, we designed a programmable AGU that has only an adder and a counter as shown in Fig. 9. The AGU creates an address at every clock cycle. We can reconfigure the AGU parameters  $m$ ,  $n$ ,  $P$ , and  $c$  to produce different addressing patterns. More complex addressing patterns can be generated by dynamic reconfiguration of AGUs and such a technique is used in the proposed MIMD-2D accelerator.

#### 3.2 SIMD 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 [11]. It has 8 PEs and a  $16 \text{ bit} \times 256$  16-bank shared memory connected through a crossbar interconnection network as shown in Fig. 10. The interconnection network is very simple and does not contain any arbiters or FIFOs. To implement and application, we have to divided it into multiple threads where 8 such threads are executed in parallel. After the execution is finished, new threads are fed. When all threads are executed, the resulting data are read by the CPU.

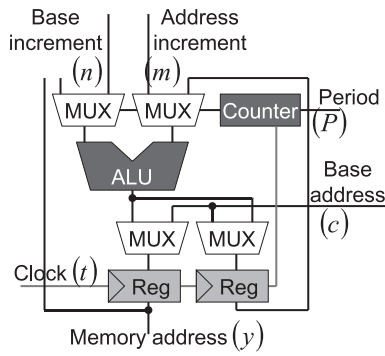


Fig. 9. Address generation unit (AGU).

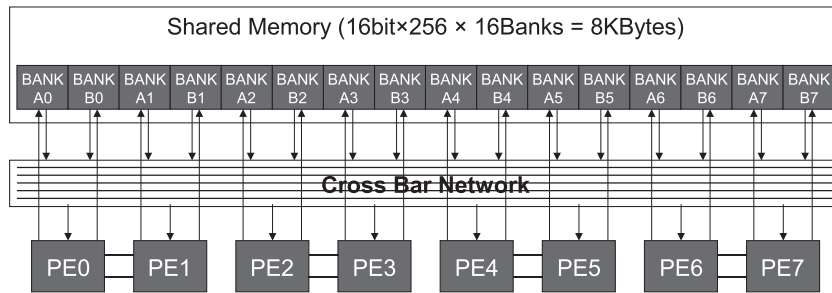


Fig. 10. SIMD-1D architecture.

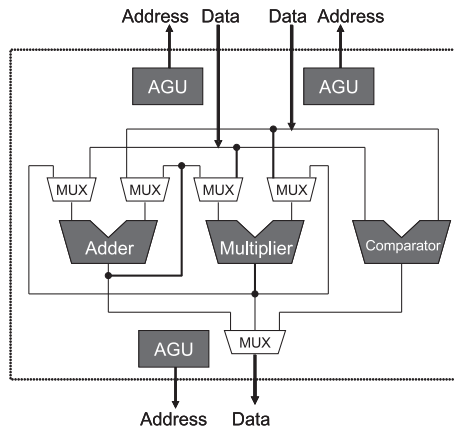


Fig. 11. PE of the SIMD-1D accelerator.

In the GPU accelerator, both the address generation and the data processing is done in ALUs. However, in the proposed SIMD-1D accelerator, the address generation is done in AGUs and the data processing is done in adders and multipliers. Therefore, we simplify the adders and multipliers to perform the most common mathematical operations power efficiently. Moreover, we can pipeline the address generation and data processing to increase the processing speed. Figure 11 shows the architecture of a PE. It consists of an adder, a multiplier and a comparator. Adder and multiplier are pipelined to perform add-multiply or multiply-add operations. Each PE has two input ports and 1 output port. A PE contains two AGUs to create addresses for the input data. However, the output data bypass the cross-bar network and directly connected to the shared memory as shown in Fig. 10. Therefore, we can reduce the area by sharing a single address generation unit among all the 8 PEs to create the output addresses. Two PEs work as a group to perform more complex operations such as “absolute difference” calculation and conditional branches.

### 3.3 MIMD accelerator

The proposed MIMD-2D accelerator is designed based on the FE-GA accelerator [1] that has a dynamically reconfigurable PE array. As shown in Fig. 12, the proposed architecture consists of 16 PEs and 6 memory modules. We use a simple interconnection network that connects PEs with their 4 nearest neighbors. To implement and application, we have to divided it into multiple contexts that execute sequentially. Within a context, we can perform parallel

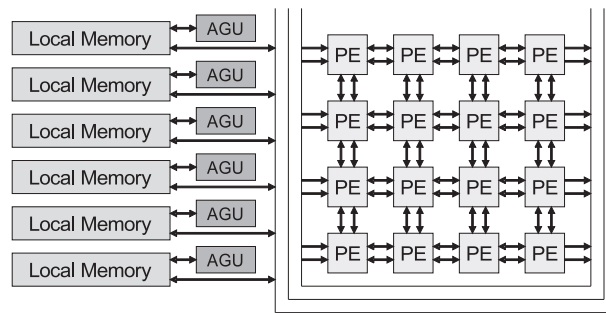


Fig. 12. MIMD-2D architecture.

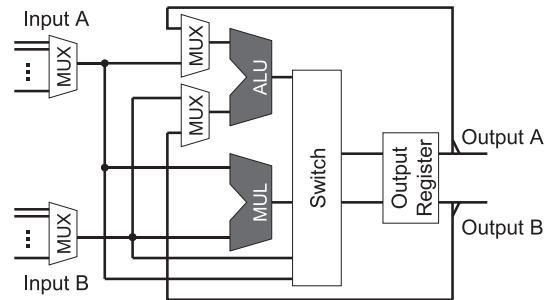


Fig. 13. PE of the MIMD-2D accelerator.

computations. The computation starts after the configuration data of multiple contexts are written to the configuration memory of the accelerator. When the computation is finished, the resulting data are read by the CPU.

Figure 13 shows the structure of a PE. It consists of a 16-bit adder and a 16-bit multiplier to perform basic mathematical operations. A single PE performs basic mathematical operations such as addition, multiplication, etc. More complex operations such as absolute difference and conditional operations are done by connecting several PEs. Moreover, it is possible to execute 32-bit or 64-bit calculations by connecting PEs.

Unlike FE-GA accelerator, the proposed MIMD-2D architecture uses dynamic partial reconfiguration of AGUs to generate complex addressing patterns. That is, some AGU parameters can be changed dynamically while the rest of the circuit is at the operational state. Partial reconfiguration reduces the configuration memory size by removing redundant configurations. It also reduces the reconfiguration time by changing only a small part of the circuit.

### 3.4 Evaluation

We implement the proposed architecture using the Altera DE3 board shown in Fig. 14 that contains a Stratix III EP3SL150F1152C2 FPGA. We used “NIOS II” 32-bit RISC processor as the CPU core. It is designed using Altera SOPC builder ver. 9.1 and programmed by “C language” using Nios II 9.1 IDE tool. The Verilog HDL code of the accelerator is generated automatically by setting the parameters such as the number of PEs, connection between PEs, number of memory modules, memory capacity of a memory module, etc.

We have evaluated each core in the proposed architecture for the number of LUTs, registers, DSP blocks, and memory bits. The number of LUTs required by SIMD-1D and MIMD-2D accelerators is around 4% of the total



Fig. 14. FPGA board (DE3 board) used for the implementation.



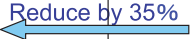
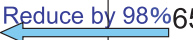
Table 2. Comparison of AGUs.

AGU type	Number of LUTs	Number of registers	Delay (ns)	Dynamic power (mW)
Proposed AGU	21	16	1.73	4.41
Used in [6]	311	47	16.39	7.16

Table 3. Comparison of the SIMD accelerator with a GPU.

	GPU (NVIDIA 8400GS)	Proposed architecture	
		1 CPU and 1 accelerator	1 CPU and 3 accelerators
Clock frequency (MHz)	900	150	150
Processing time (ms)	368	895	299
Power consumption (W)	38	1	1.2

Table 4. Comparison of the MIMD-2D accelerator with a CPU.

Processors	Proposed architecture	CPU (for reference)
	1CPU + 2 Accelerators (150MHz)	Core2 Duo E6850 (3GHz)
Processing time	22ms  Reduce by 35%	34ms
Power	1.3W  Reduce by 98%	65W(TDP)

LUTs available in the FPGA. Therefore, it is possible to implement up to 24 accelerator cores in Stratix III EP3SL150F1152C2 FPGA.

Table 2 shows the area, power consumption and delay of the proposed simple AGU compared to a complex AGU used in [6]. According to the results, the area, delay and power consumption is reduced by 14.8, 9.4, and 1.6 times respectively. Therefore, we can say that the proposed AGU is very compact and power-efficient.

Table 3 shows the power-efficiency of the proposed SIMD-1D accelerator compared to the NVIDIA 8400GS GPU. We used matrix multiplication for the evaluation and the matrix size is  $1024 \times 1024$ . According to the results on Table 3, the processing time of the proposed architecture with one accelerator is large compared to that of the GPU. However, we can decrease the processing time by using three accelerators. The power consumption in the proposed architecture is very small and the power-efficiency is more than 15 times compared to the GPU. We can increase the processing speed and power-efficiency further by using more accelerator cores.

Table 4 shows the comparison of the FPGA-based platform with the MIMD-2D accelerator cores. By using only two accelerator cores, the processing time is reduced by 35%, and the power consumption by 98%.

## 4. Conclusion

To solve the data-transfer bottleneck between different cores in a heterogeneous multi-core processor, memory allocation and task allocation are presented. The proposed memory allocation can reduce the data transfers by more than 95% to the conventional one since it allows to reuse data in the local memories of the accelerator. Moreover, the proposed task allocation can reduce by about 70% respectively compared to the conventional one since it maps consecutive tasks onto the accelerator.

For further reduction of the data-transfer amount, we are currently developing new technologies in architectural level such as a data-compression-based approach. Moreover, an FPGA-based platform with SIMD/MIMD accelerator cores is presented. To enhance the speed and power-efficiency, we are developing automatic data-path optimization of the accelerator cores.

## REFERENCES

- [1] Shikano, H., Ito, M., Onouchi, M., Todaka, T., Tsunoda, T., Kodama, T., Uchiyama, K., Odaka, T., Kamei, T., Nagahama, E., Kusaoka, M., Nitta, Y., Wada, Y., Kimura, K., and Kasahara, H., "Heterogeneous multi-core architecture that enables 54x AAC-LC stereo encoding," *IEEE Journal of Solid-State Circuits*, **43**: 902–910 (2008).
- [2] Kondo, H., Nakajima, M., Masui, N., Otani, S., Okumura, N., Takata, Y., Nasu, T., Takata, H., Higuchi, T., Sakugawa, M., Fujiwara, H., Ishida, K., Ishimi, K., Kaneko, S., Itoh, T., Sato, M., Yamamoto, O., and Arimoto, K., "Design and implementation of a configurable heterogeneous multicore SoC with nine CPUs and two matrix processors," *IEEE Journal of*

- Solid-State Circuits*, **43**: 892–901 (2008).
- [3] <http://www.top500.org/system/10587>
  - [4] Dalal, N., and Triggs, B., “Histograms of oriented gradients for human detection,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 886–893 (2005).
  - [5] Waidyasooriya, H. M., Hariyama, M., and Kameyama, M., “Memory allocation for window-based image processing on multiple memory modules with simple addressing functions,” *IEICE Transaction on Fundamentals*, **E94-A**: 342–351 (2011).
  - [6] Kobayashi, Y., Hariyama, M., and Kameyama, M., “Optimal periodic memory allocation for image processing with multiple windows,” *IEEE Trans. VLSI Systems*, **17**: 403–416 (2009).
  - [7] Waidyasooriya, H. M., Ohbayashi, Y., Hariyama, M., and Kameyama, M., “Memory allocation exploiting temporal locality for reducing data-transfer bottlenecks in heterogeneous multicore processors,” *IEEE Transactions on Circuits and Systems for Video Technology*, **21**: 1453–1466 (2011).
  - [8] Waidyasooriya, H. M., Hariyama, M., and Kameyama, M., “Task allocation with algorithm transformation for reducing data-transfer bottlenecks in heterogeneous multi-core processors: A case study of HOG descriptor computation,” *IEICE Transaction on Fundamentals*, **E93-A**: No. 12 (2010).
  - [9] Volder, J. E., “The CORDIC trigonometric computing technique,” *IRE Transactions on Electronic Computers*, 330–334 (1959).
  - [10] Waidyasooriya, H. M., Takei, Y., Hariyama, M., and Kameyama, M., “FPGA implementation of heterogeneous multicore platform with SIMD/MIMD custom accelerators,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, 1339–1342 (2012).
  - [11] Waidyasooriya, H. M., Hariyama, M., and Kameyama, M., “Architecture of an FPGA-oriented heterogeneous multi-core processor with SIMD-accelerator cores,” *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 179–186 (2010).