IEICE TRANS. ELECTRON., VOL.E95–C, NO.12 DECEMBER 2012

PAPER

Acceleration of Block Matching on a Low-Power Heterogeneous Multi-Core Processor Based on DTU Data-Transfer with Data Re-Allocation

Yoshitaka HIRAMATSU[†a)], **Hasitha Muthumala WAIDYASOORIYA**[††], **Masanori HARIYAMA**[††], **Toru NOJIRI**[†], ***Members***, **Kunio UCHIYAMA**[†], ***and*** **Michitaka KAMEYAMA**[††], ***Fellows***

SUMMARY The large data-transfer time among different cores is a big problem in heterogeneous multi-core processors. This paper presents a method to accelerate the data transfers exploiting data-transfer-units together with complex memory allocation. We used block matching, which is very common in image processing, to evaluate our technique. The proposed method reduces the data-transfer time by more than 42% compared to the earlier works that use CPU-based data transfers. Moreover, the total processing time is only 15 ms for a VGA image with 16×16 pixel blocks.

key words: *block matching, heterogeneous multi-core, dynamically reconfigurable processor, data transfer, accelerator*

1. Introduction

Today's digital appliances such as mobile phones, TVs, and digital cameras require real-time image processing at low-power. However, the power consumption of conventional CPU-based image processing systems is very high. Therefore, an effective way to implement image processing is to use low-power heterogeneous multi-core processors that contain different cores such as CPUs and accelerators. Examples of heterogeneous multi-core processors are [1] and [2]. In heterogeneous multi-core processors, different tasks of an application are assigned to the most suitable processor core. Then several cores work collectively to improve the overall performance. Moreover, heterogeneous multi-core processors are programmable by software, so that the design cost and time are significantly low.

In this paper, we target the heterogeneous multi-core processor called RP-X, we previously proposed for digital appliances [3]. It has SH-4A CPU cores and FE-GA (flexible engine/generic ALU array) accelerator cores. To reduce the power consumption and to increase the processing speed, the accelerators in this processor contain a hierarchical memory structure, a small number of processing elements (PEs) and address generation units (AGUs) as shown in Fig. 1. The hierarchical memory structure contains a large memory module (global memory) placed outside the accelerator and several small memories (local memories) placed inside the accelerator. Local memories provide high-speed and parallel data access at low power. However, their memory capacity is very small. Therefore, we have a large global memory outside the accelerator. RP-1 [1] is another heterogeneous multi-core processor that has this memory structure. Data transfer units (DTUs) are included to accelerate the linear and stride data-transfers between the global and local memories. The accelerator cores employ AGUs for fast address generation. To decrease the area of the accelerator, the AGUs contain simple hardware units such as adders and counters. Therefore the AGUs implement only the addressing functions of the simple memory access patterns [4] such as linear and stride accesses. Due to this addressing function constraint, the same data have to be copied many times and it is called the "data duplication problem". Figure 2 shows this problem. Figures 2(a) and 2(b) show the coordinates of the pixels of an image and the control steps where a set of pixels are accessed respectively. To access these pixels, we use a simple addressing function. Figure 2(c) shows one possible memory allocation. In this example, the pixel [0,1] is copied to two memory locations: 0x01 and 0x05. Similarly, the pixel [1,2] is copied to 0x04 and 0x08. Even though we need to access only 8 pixels, we have to transfer 10 pixels to the local memory modules where two of them are duplicated.

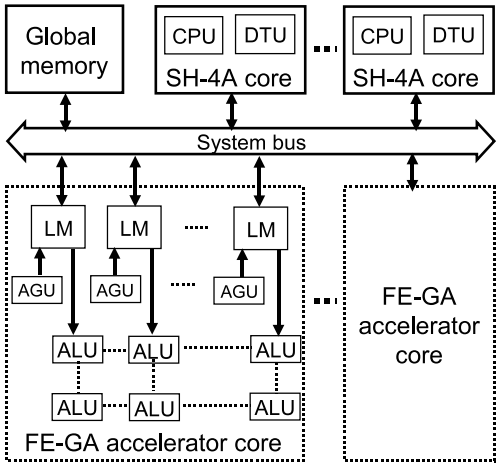


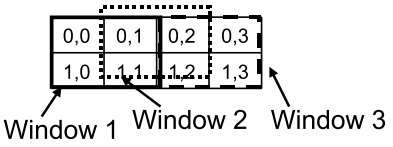
Fig. 1 RP-X heterogeneous multi-core processor architecture.

To solve the data duplication problem, the memory al-

Manuscript received February 23, 2012.
Manuscript revised July 5, 2012.

[†]The authors are with Central Research Laboratory, Hitachi, Ltd., Kokubunji-shi, 185-8601 Japan.
[††]The authors are with the Graduate School of Information Sciences, Tohoku University, Sendai-shi, 980-8579 Japan.
a) E-mail: yoshitaka.hiramatsu.xw@hitachi.com

DOI: 10.1587/transele.E95.C.1872



(a) Coordinates of the pixel data

Control Step (t)	0~3	4~7	8~11
Access window	Window 1	Window 2	Window 3
data	0,0 0,1 1,0 1,1	0,1 0,2 1,1 1,2	0,2 0,3 1,2 1,3

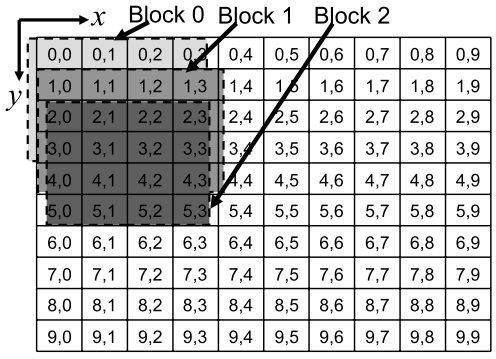
(b) Data access sequence.

Memory address (A)	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09
data	0,0	0,1	1,0	1,1	1,2	0,1	0,2	0,3	1,2	1,3

Addressing function used to access data $A = t - \left\lfloor \frac{t}{4} \right\rfloor$

(c) Memory allocation example.

Fig. 2 Data-duplication due to the addressing function constraint.



0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9

(a) Coordinates of the pixels in the candidate image

Address	00	01	02	03	04	05	06	07	08	09	10	11
Memory 0	0,0	2,0	0,1	2,1	0,2	2,2	0,3	2,3	0,4	2,4	0,5	
Memory 1	1,0	3,0	1,1	3,1	1,2	3,2	1,3	3,3	1,4	3,4	1,5	

Address	...	20	21	22	23	24	25	26	27	28	31
Memory 0		2,0	4,0	2,1	4,1	2,2	4,2	2,3	4,3	2,4	

(b) Allocated pixel data on memory modules

Fig. 3 Memory allocation for horizontal-first schedule.

location method based on data sharing is proposed in [5]. In this method, the data are allocated to the local memories in such a way that they can be accessed using simple addressing functions. However, complex data-transfers are required to implement this memory allocation. Such complex data-transfers cannot be accelerated using the DTU. As a result, the data-transfer time is usually large and the processing time is taken up by the data-transfers.

In this paper, we propose a DTU-based data-transfer and data re-allocation method to obtain the same memory allocation result in [5] much faster. Initially, the data transfer to the local memories is accelerated using the DTU. Then, the data are re-allocated in the local memories so as to obtain the memory allocation result in [5]. The data re-allocation overhead is very small since it is done in parallel using multiple AGUs. To verify the effectiveness of this method, we use a block matching example. Block matching is widely used in many image processing applications such as stereo vision [6], optical-flow extraction [7], etc. According to the results, the proposed method reduces the data-transfer time by more than 42% compared to that in [5].

2. Previous Works

Much research have been done already on memory allocation in previous works such as [6], [8], [9]. A hierarchical matching approach for stereo matching to reduce the computation amount is proposed in [6]. The parallel access of multiple memory modules is discussed in [8] and [9]. These methods are proposed under the assumption that the random memory access is possible and the allocated data are accessible at any time from any memory address.

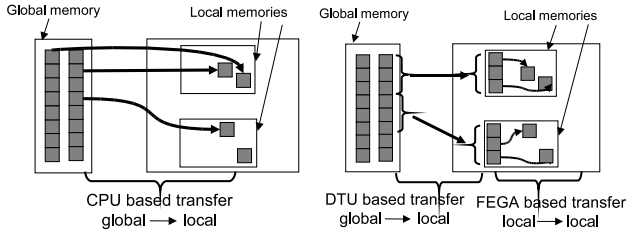
In this work, we focus on RP-X heterogeneous multi-core processor [3] that we developed for digital appliances. The types of cores in the RP-X processor are the same as those in the RP-1 processor that we proposed in our previous work [1]. The difference between these two processor architectures is the number of cores. RP-1 has four SH-4A CPU cores and two FE-GA accelerator cores while RP-X has 8 SH-4A cores and four FE-GA cores. A detailed description of RP-1 and RP-X is given in [10] on chapters 4.2 and 4.4 respectively. To reduce the power consumption, the accelerators in these processors have a small number of reconfigurable PEs. If we use these small number of PEs for the address generation, we cannot use them for data processing and this reduces the processing speed. Therefore, it is not efficient to use PEs to generate the memory addresses required for the random memory access. To solve the problem, the accelerator has special hardware units called AGUs for the address generation. AGUs contain simple hardware units such as adders and counters to reduce the accelerator core area. AGUs generate addresses for the simple and most common memory access patterns in media processing such as linear and stride access. However, AGUs cannot generates addresses for more complex and irregular memory access patterns such as random access. Therefore, the traditional memory allocation techniques cannot be applied.

For such heterogeneous multi-core processors, addressing-function-constrained memory allocation is proposed in [5]. Figure 3 shows an example of this memory allocation. Figure 3(a) shows the coordinates of the pixel data in the scan area. The scan area width and the height are 10 and 10 respectively. A block of size 4×4 is used for the scanning. Two memory modules are used to allocate pixel data. Figure 3(b) shows the allocated data on memory modules. The data duplication is reduced by sharing the data among horizontal-blocks. As shown in Fig. 3(b), the data in memory module 1 are shared between the blocks 0 and 1. Similarly, the data in module 0 are shared between the blocks 1 and 2 and so on. However, there is a small data duplication. For example, pixel $(2, 1)$ is allocated to the memory

addresses 03 and 22 of memory module 0.

To implement this memory allocation, we have to transfer data from the global memory (source) to the local memories (destination). Such data transfers take a large processing time and need to be accelerated. DTU is very efficient way of accelerating burst-mode data transfers where a large segment of data are transferred from one memory location to another. Usually, both the throughput and the latency of the DTU-based data transfer is large compared to the CPU-based data transfer. Therefore, larger data segments give more efficient data transfers. However, the memory allocation in [5] is very complicated and we cannot transfer a one large segment of data from the source to the destination. If we use DTU to realize this memory allocation, the data block size becomes very small. For such small block sizes, the data transfer cannot be accelerated. Therefore, CPU-based data-transfer is used in [5].

In this paper, we propose a DTU-based data transfer acceleration method with data re-allocation to obtain the same memory allocation result in [5]. Figure 4 shows the difference between the proposed method and the method in [5]. As shown in Fig. 4(a), the method in [5] transfers one data at a time from the global memory to the local memories using the CPU. In the proposed method shown in Fig. 4(b), we divide the data transfer into two steps. In the first step, a large segment of data in the global memory is transferred to the local memory using the DTU. In the second step, the data in the local memory are re-allocated to different addresses in the same local memory to achieve the memory allocation result in [5]. Since we transfer the same data twice, global to local and local to local, it looks like we are wasting processing time. However, as shown in Fig. 5, the data re-allocation overhead is very small compared to the time reduction due to DTU-based data-transfer. We achieved such a small data re-allocation time by employing parallel local-to-local data transfers using multiple AGUs.



(a) Data-transfer method in [5] (b) Proposed data transfer method

Fig. 4 Conventional vs. proposed data-transfers.

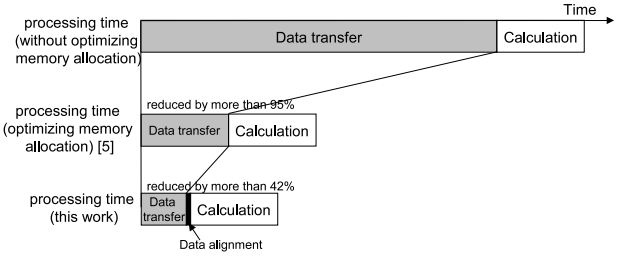


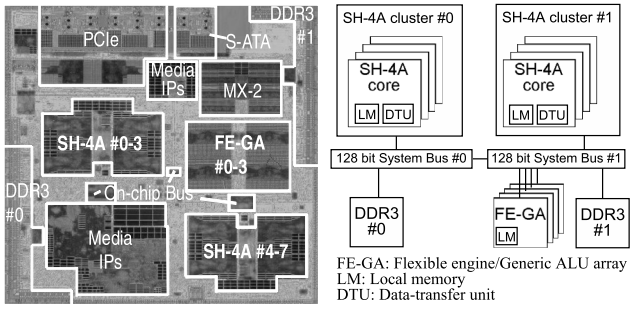
Fig. 5 Data-transfer time and total processing time.

3. Heterogeneous Multi-Core Processor Architecture

In this paper, we use the heterogeneous multi-core chip (RP-X) [3] we previously developed for digital appliances. A micrograph of this chip is shown in Fig. 6(a). A block diagram of the chip is shown in Fig. 6(b). It has four types of processors: eight SH-4A cores (two SH-4A clusters, each of which is composed of four SH-4A cores), four FE-GAs [1], two MX-2s [11], and one video-processing unit 5 (VPU5) [12], [13]. In this research, we use four SH-4A cores in one cluster and four FE-GAs cores. Each SH-4A core is a reduced instruction set computer (RISC) processor.

3.1 Data-Transfer Module

The heterogeneous multi-core processor [3] has data-transfer modules called Data transfer units (DTUs). The data-transfer latency and throughput of the SH-4A and DTU are listed in Table 1. The values are based on the data transfer between the SH-4A local memory and FE-GA local memory. The latency of the SH-4A is referred to the number of CPU-clock-cycles between the start-time of reading data from the SH-4A local memory and the end-time of writing it to the FE-GA local memory. The latency of the DTU for the data transfer between SH-4A local memory and FE-GA local memories is referred to the number of CPU-clock-cycles required between the starting of the DTU and the end-time of writing the first data to the FE-GA local memory. The throughput of the SH-4A transfer is referred to the number of byte per CPU-clock-cycle while the data are transferred in 4 bytes from the SH-4A local memory to the FE-GA local memory. The throughput of DTU is referred to the number of byte per CPU-clock-cycle while the data are transferred in



(a) Chip micrograph (b) Block diagram

Fig. 6 Heterogeneous multi-core chip proposed in [3].

Table 1 Data transfer latency and throughput.

Processor core	Latency (CPU clock)	Throughput (B/CPU clock)
SH-4A	38	0.50
DTU	50	0.67

(CPU frequency: 648 MHz; system bus frequency: 324 MHz)

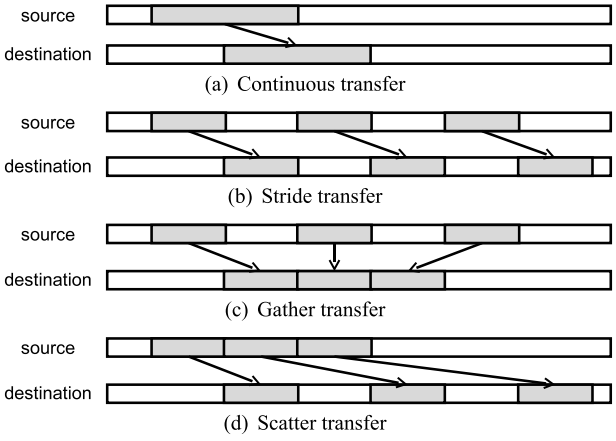


Fig. 7 DTU commands.

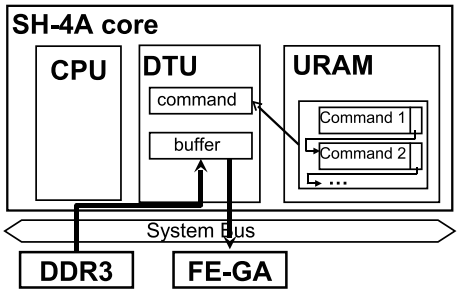


Fig. 8 Data flow between DDR3 and FE-GA.

16 bytes from the SH-4A local memory to the FE-GA local memory. According to the results in Table 1, data transfer using DTUs is faster than the data transfer using CPU.

The DTU can be programmed by placing transfer commands on the SH-4A local memory. Four types of commands are used: continuous transfer, stride transfer, gather transfer, and scatter transfer. Each command is shown schematically in Fig. 7. The data flow between DDR3-SDRAM and FE-GA using DTU is also shown in Fig. 8. The DTU reads commands from the SH-4A local memory, and it reads from the DDR3-SDRAM through a system bus to the DTU local memory and writes to the FE-GA local memory through the system bus. The commands can also be placed as a linked list in the SH-4A local memory which is called the URAM. Then the DTU reads commands from the URAM and executes one-by-one as shown in Fig. 8.

3.2 Flexible Engine/Generic ALU Array

The FE-GA [1] is a non-uniformed processing element array and a dynamically reconfigurable processor. A block diagram of the FE-GA is shown in Fig. 9. It consists of thirty-two 16-bit processing element cells, ten load store cells (LSs), ten 4 KB local memory cells (CRAMs), a configuration manager (CFGM), a sequence manager (SEQM), and a crossbar network (XB), which contains two kinds of PEs. One kind consists of an arithmetic logic unit (ALU), a shifter, and registers. There are twenty-four of these elements. The other kind consists of multiply accumulation units (MLTs) and registers, and there are eight of these elements. The PEs are arrayed two-dimensionally and connected to neighboring cells. The CFGM is programmable and can change the type and connection of PEs and the XB connection during certain clock cycles. The FE-GA contains 256 sequences which are dynamically reconfigurable. Therefore, in each sequence, we can change the operations in ALU, MLT, LS cells and their connections. We also can change the connections in the crossbar network. The SEQM performs autonomous sequence control, creating a highly independent subsystem. The FE-GA is suitable for middle-grained operations with middle parallelism. In particular, it accelerates image processing including many multiple accumulation operations such as finite impulse response (FIR).

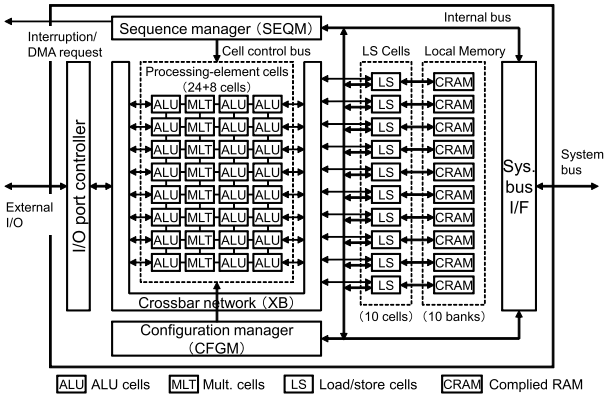


Fig. 9 Block diagram of FE-GA.

3.3 Address Generation in FE-GA

The FE-GA has AGUs placed inside the LS (load/store) cells as shown in Fig. 9 for address generation. The address generation using AGUs is very useful since it significantly decreases the address calculation time. It also allows ALU and MLT cells to concentrate only on data processing. To reduce the area of the accelerator core, AGUs contain only simple hardware such as adders and counters. Therefore, the number addressing patterns generated in AGUs are limited to most common addressing patterns. The relationship between the time and the control step (clock cycle) is called an "addressing function". In FE-GA, the addressing functions are limited to linear functions as shown in Eq. (1).

$$Address = m \times t + c \tag{1}$$

The parameters m, t and c are the *address increment*, the *control step* and the *base address* respectively. There is another parameter called *number of iterations* that determines how many clock cycles this addressing function works. After the addressing function works by the *number of iterations*, address returns to the *base address* as shown in Fig. 10. In each context of the FE-GA, we need to set these 3 parameters m, c and the *number of iterations*. Therefore, it is possible to change those parameters dynamically to ac-

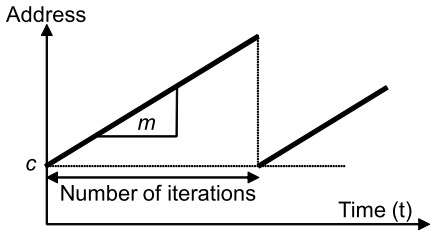


Fig. 10 Addressing function.

cess different parts of the memory.

4. Block-Matching Computation on Heterogeneous Multi-core Processor

4.1 Block Matching

In this paper, we consider the data transfer in the optical-flow extraction application based on block matching. In the block matching, corresponding pixels between two images taken at time t and $t + \delta t$ are searched. To find the corresponding pixel, a reference block for a particular pixel in the image at time t and a search area in the image at time $t + \delta t$ are considered as shown in Fig. 11. Different candidate blocks are selected from the search area and the SAD (sum of absolute differences) value with the reference block is calculated. The SAD is calculated using Eq. (2) where N, M, $f(x, y)$ and $g(x, y)$ are the width of a block, the height of a block, a pixel in the image at time t (reference image) and a pixel in the image at time $t + \delta t$ (candidate image) respectively.

$$S_{sad} = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} |f(x, y) - g(x, y)| \tag{2}$$

The similar the reference block to the candidate block is, the smaller the SAD becomes. Therefore, the candidate block with the minimum SAD value is selected as the corresponding block to the reference block. The specifications of the block matching are given in Table 2.

4.2 Implementation

The block matching algorithm used in [5] is implemented in this paper using the proposed data transfer method. As shown in Fig. 12(a), the candidate block is moved one pixel from left-to-right and up-to-down in the search area. The pixels inside a block are accessed in columns from left-to-right as shown in Fig. 12(b). The pixels in a column are stored in multiple memory modules and accessed in parallel. This scheduling is called block-serial-pixel-parallel scheduling. This scheduling is suitable for the FE-GA since the partial SAD calculation for the pixels in a column can be easily mapped onto the mesh-connected cells as described in Sect. 4.2.3. A detailed description on the access order of the pixels is given in [5].

The block matching contains two major tasks; SAD calculation and searching for the minimum SAD. The SAD calculation takes more than 99% of the total processing time [5]. Therefore, we use the FE-GA for the SAD calculation. The SH-4A is used for the minimum SAD search since it contains a large amount of control processing. A pair composed of an SH-4A and an FE-GA executes the SAD calculation and the search for minimum SAD respectively. Four

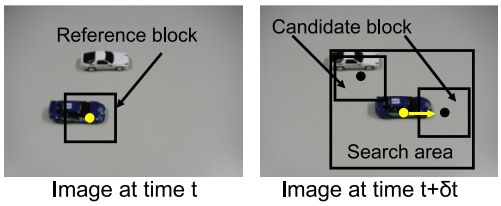
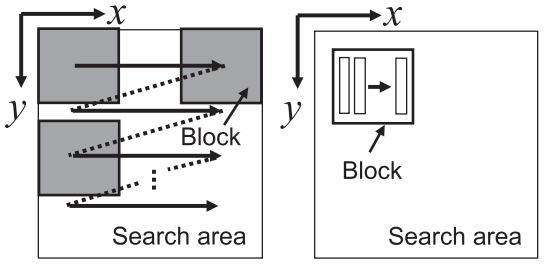


Fig. 11 Optical-flow extraction based on block matching.

Table 2 Specification of the block matching.

Image size	640 × 480
Search area size	24 × 24
Block size	16 × 16



(a) Movement of the blocks (b) Access sequence for the pixels in a block

Fig. 12 Access order of pixels.

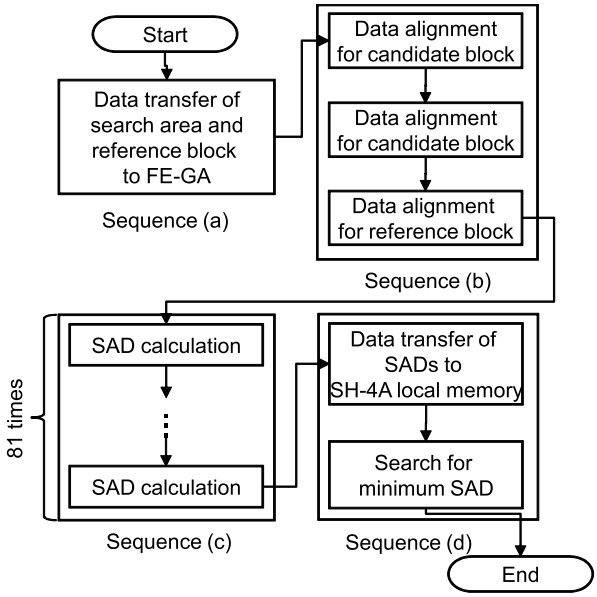


Fig. 13 Flow-chart of block matching on a SH-4A/FE-GA pair.

Address	0	1	2	...	639	640	641	...	1279	1280	...	307199	307200	...	614399
Data	0,0	0,1	0,2	...	0,639	1,0	1,1	...	1,639	2,0	...	479,639	0,0	...	479,639

Candidate image data — Reference image data

Fig. 14 Pixel data stored in the DDR3-SDRAM.

y \ x	0	1	2	...	15	16	17	...	...	21	22	23	24	...
0	0,0	0,1	0,2	...	0,15	0,16	0,17	...	...	0,21	0,22	0,23	0,24	...
1	1,0	1,1	1,2	...	1,15	1,16	1,17	...	...	1,21	1,22	1,23	1,24	...
2	2,0	2,1	2,2	...	2,15	2,16	2,17	...	...	2,21	2,22	2,23	2,24	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
15	15,0	15,1	15,2	...	15,15	15,16	15,17	...	...	15,21	15,22	15,23	15,24	...
16	16,0	16,1	16,2	...	16,15	16,16	16,17	...	...	16,21	16,22	16,23	16,24	...
17	17,0	17,1	17,2	...	17,15	17,16	17,17	...	...	17,21	17,22	17,23	17,24	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21	21,0	21,1	21,2	...	21,15	21,16	21,17	...	...	21,21	21,22	21,23	21,24	...
22	22,0	22,1	22,2	...	22,15	22,16	22,17	...	...	22,21	22,22	22,23	22,24	...
23	23,0	23,1	23,2	...	23,15	23,16	23,17	...	...	23,21	23,22	23,23	23,24	...
24	24,0	24,1	24,2	...	24,15	24,16	24,17	...	...	24,21	24,22	24,23	24,24	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Candidate blocks — Search area

Fig. 15 Coordinates of the pixel data in the image.

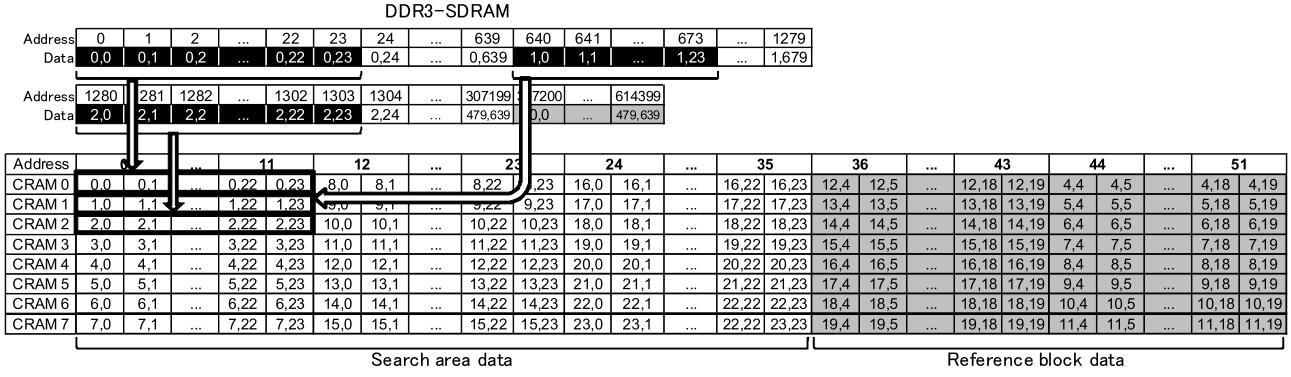


Fig. 16 Data transferred to the CRAMs.

such pairs are used for the block matching where each processes 25% of the blocks in the image. A flow chart of the block matching by the SH-4A/FE-GA pair is shown in Fig. 13. In sequence (a), the SH-4A continuously transfers the candidate blocks and reference blocks to the FE-GA local memory. In sequence (b), the FE-GA aligns the candidate block and the reference block in its own local memory. In sequence (c), the FE-GA calculates SAD 81 times by picking different candidate blocks on the search area. In sequence (d), the SH-4A reads these SADs from the FE-GA local memory and searches for the minimum SAD. The following section describes the flow from sequences (a) to (d).

4.2.1 Data-Transfer to FE-GA

The data transfer from DDR3-SDRAM to the CRAMs in FE-GA is done by the DTU to decrease the processing time. Figure 14 shows the data stored in the DDR3-SDRAM. Note that the data are represented by the coordinates of the pixels as shown in Fig. 15. The data are stored in DDR3-SDRAM from line-by-line as in the raster scan. The data of the candidate image is stored from the addresses 0 to 307199. Then the data of the reference image are stored in addresses 307200 to 614399.

Figure 16 shows the data transferred from the DDR3-SDRAM to 8 CRAMs in the FE-GA for one search area and one reference block. Since the CRAMs are too small to hold all the data in two images, a reference block and its corresponding search area are transferred one at a time. After the SAD computation is finished, another reference block and a search area are transferred. Since the search area size is 24×24, the first search area contains the data (0,0) ~ (0,23), (1,0) ~ (1,23), ... , (23,0) ~ (23,23) as shown in Fig. 15. The data: (0,0) ~ (0,23), (1,0) ~ (1,23), ... , (7,0) ~ (7,23) are stored from CRAM 0 to CRAM 7 respectively. Then the data: (8,0) ~ (8,23), ... , (15,0) ~ (15,23) are stored

from CRAM0 to CRAM 7 respectively. Similarly the rest of the data are stored as shown in Fig. 16. Since the word length of the DDR3-SDRAM and the CRAMs are 8 bits and 16 bits respectively, two pixels are stored in each CRAM address. The CRAM address from 0 to 35 contain the search area data of the candidate image. The addresses from 36 to 51 contain the reference block data of the reference image.

To transfer data from the DDR3-SDRAM to CRAMs, we use the stride transfer mode of the DTU explained in Fig. 7(b) where the source is the DDR3-SDRAM and the destinations are the CRAMs. Although there are several CRAMs, their addresses are mapped to a global address space so that we can see them as a single memory. The data are transferred by using a command-list of five stride transfer commands shown in Fig. 17 where each of which has different source and destination addresses. For example, the data transfer using command 1 is shown in Fig. 18. In this command, the source address is 0 since the pixel coordinate (0,0) is stored in address 0 of DDR3-SDRAM. The destination address is 0 since the pixel (0,0) is stored in the first address of CRAM 0. The stride width is 24. The gap between two strides is 616 and 2036 for source and destination respectively.

4.2.2 Re-Allocation of Data Using FE-GA

In the proposed method, we re-align the data in the CRAMs to obtain the memory allocation result in [5]. As explained in Sec.4.2.1, two 8 bit pixels are stored in one CRAM address. First, we separate the pixels and then transfer those to different CRAM addresses. This process is called the "data re-allocation". The data re-allocation is divided into 3 simple phases and each phase is done in 2 sequences.

In this paragraph, we explain how the data re-allocation is done using FE-GA. Figure 19 shows the PE array of the FE-GA that do the re-allocation. In the first sequence, the upper 8 bit of data in CRAM addresses (the first pixel) are extracted and converted to a 16 bit data by adding zeros as shown in Fig. 19(a). After the bit conversion, the data are transferred to another address in the same CRAM. Similarly, the lower 8 bit of data (the second pixel) are extracted and transferred in the second sequence as shown in Fig. 19(b). Eight such data transfers are done in parallel and their addresses are generated in AGUs.

This paragraph explains the different data transfer in phases 1 to 3. Figure 20 shows the data transfer of the phase 1. The addresses 0 to 51 are the source addresses (same as Fig. 16) and the addresses from α to α + 111 are the destination addresses. (Note that the data shown in gray background are the reference block data). The value α is an offset that separates the read and write address spaces. In the first sequence, the data in source addresses: 0 to 23 are read.

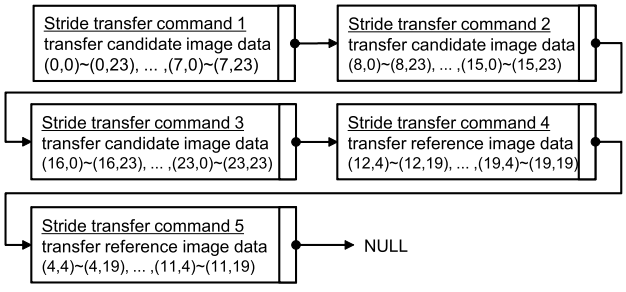


Fig. 17 DTU command lists for data transfer in sequence (a).

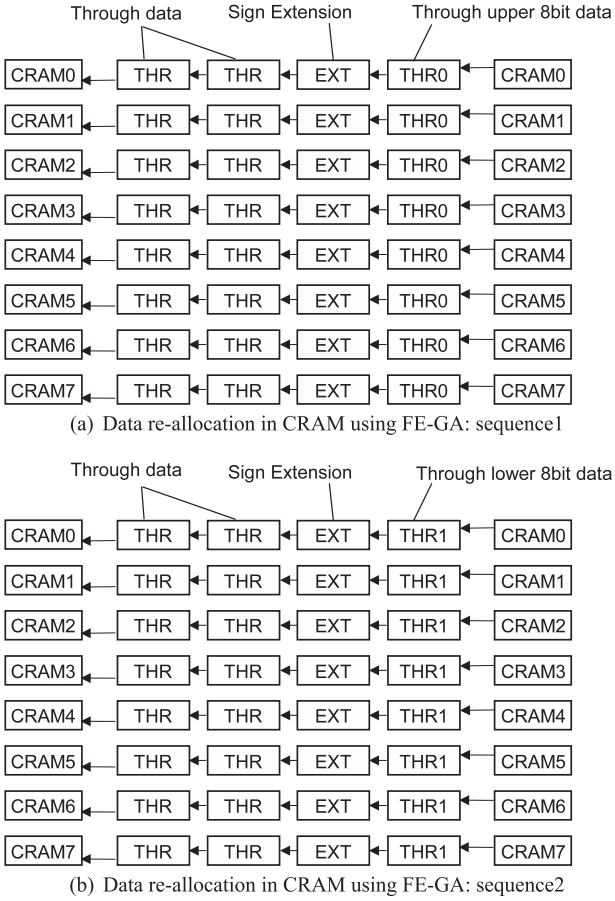


Fig. 19 Data re-allocation in CRAM using FE-GA.

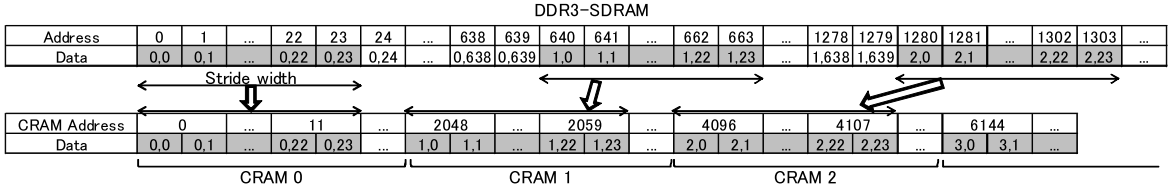


Fig. 18 Stride data transfer of DTU command 1 in Fig. 17.

Address	0	1	2	...	51	...	α+0	α+1	α+2	α+3	α+4	...	α+44	α+45	α+46	α+47	α+48	α+49	α+50	...	α+94	...
CRAM 0	**0,0** 0,1	**0,2** 0,3	0,4 0,5	...	4,18 4,19	...	***0,0***				***0,2***	...	***0,22***				***8,0***			...		
CRAM 1	1,0 1,1	1,2 1,3	1,4 1,5	...	5,18 5,19	...	***1,0***				***1,2***	...	***1,22***				***9,0***			...		
CRAM 2	2,0 2,1	2,2 2,3	2,4 2,5	...	6,18 6,19	...	***2,0***				***2,2***	...	***2,22***				***10,0***			...		
CRAM 3	3,0 3,1	3,2 3,3	3,4 3,5	...	7,18 7,19	...	***3,0***				***3,2***	...	***3,22***				***11,0***			...		
CRAM 4	4,0 4,1	4,2 4,3	4,4 4,5	...	8,18 8,19	...	***4,0***				***4,2***	...	***4,22***				***12,0***			...		
CRAM 5	5,0 5,1	5,2 5,3	5,4 5,5	...	9,18 9,19	...	***5,0***				***5,2***	...	***5,22***				***13,0***			...		
CRAM 6	6,0 6,1	6,2 6,3	6,4 6,5	...	10,18 10,19	...	***6,0***				***6,2***	...	***6,22***				***14,0***			...		
CRAM 7	7,0 7,1	7,2 7,3	7,4 7,5	...	11,18 11,19	...	***7,0***				***7,2***	...	***7,22***				***15,0***			...		

Source data — Destination data

(a) Sequence 1

Address	0	1	2	...	51	...	α+0	α+1	α+2	α+3	α+4	...	α+44	α+45	α+46	α+47	α+48	α+49	α+50	...	α+94	...
CRAM 0	0,0 **0,1**	0,2 0,3	0,4 0,5	...	4,18 4,19	...	0,0		***0,1***		0,2	...	0,22		***0,23***		8,0		***8,1***	...	***8,23***	
CRAM 1	1,0 1,1	1,2 1,3	1,4 1,5	...	5,18 5,19	...	1,0		***1,1***		1,2	...	1,22		***1,23***		9,0		***9,1***	...	***9,23***	
CRAM 2	2,0 2,1	2,2 2,3	2,4 2,5	...	6,18 6,19	...	2,0		***2,1***		2,2	...	2,22		***2,23***		10,0		***10,1***	...	***10,23***	
CRAM 3	3,0 3,1	3,2 3,3	3,4 3,5	...	7,18 7,19	...	3,0		***3,1***		3,2	...	3,22		***3,23***		11,0		***11,1***	...	***11,23***	
CRAM 4	4,0 4,1	4,2 4,3	4,4 4,5	...	8,18 8,19	...	4,0		***4,1***		4,2	...	4,22		***4,23***		12,0		***12,1***	...	***12,23***	
CRAM 5	5,0 5,1	5,2 5,3	5,4 5,5	...	9,18 9,19	...	5,0		***5,1***		5,2	...	5,22		***5,23***		13,0		***13,1***	...	***13,23***	
CRAM 6	6,0 6,1	6,2 6,3	6,4 6,5	...	10,18 10,19	...	6,0		***6,1***		6,2	...	6,22		***6,23***		14,0		***14,1***	...	***14,23***	
CRAM 7	7,0 7,1	7,2 7,3	7,4 7,5	...	11,18 11,19	...	7,0		***7,1***		7,2	...	7,22		***7,23***		15,0		***15,1***	...	***15,23***	

Source data — Destination data

(b) Sequence 2

Fig. 20 Data re-allocation in phase 1.

Address	α+0	α+1	α+2	α+3	...	α+46	α+47	α+48	α+49	α+50	α+51	...	α+94	α+95	α+96	α+97	α+98	α+99	...	α+110	α+111
CRAM 0	0,0	***8,0***	0,1	***8,1***	...	0,23	***8,23***	8,0	***16,0***	8,1	***16,1***	...	8,23	***16,23***		***12,4***		***12,5***			***12,19***
CRAM 1	1,0	***9,0***	1,1	***9,1***	...	1,23	***9,23***	9,0	***17,0***	9,1	***17,1***	...	9,23	***17,23***		***13,4***		***13,5***			***13,19***
CRAM 2	2,0	***10,0***	2,1	***10,1***	...	2,23	***10,23***	10,0	***18,0***	10,1	***18,1***	...	10,23	***18,23***		***14,4***		***14,5***			***14,19***
CRAM 3	3,0	***11,0***	3,1	***11,1***	...	3,23	***11,23***	11,0	***19,0***	11,1	***19,1***	...	11,23	***19,23***		***15,4***		***15,5***			***15,19***
CRAM 4	4,0	***12,0***	4,1	***12,1***	...	4,23	***12,23***	12,0	***20,0***	12,1	***20,1***	...	12,23	***20,23***		***16,4***		***16,5***			***16,19***
CRAM 5	5,0	***13,0***	5,1	***13,1***	...	5,23	***13,23***	13,0	***21,0***	13,1	***21,1***	...	13,23	***21,23***		***17,4***		***17,5***			***17,19***
CRAM 6	6,0	***14,0***	6,1	***14,1***	...	6,23	***14,23***	14,0	***22,0***	14,1	***22,1***	...	14,23	***22,23***		***18,4***		***18,5***			***18,19***
CRAM 7	7,0	***15,0***	7,1	***15,1***	...	7,23	***15,23***	15,0	***23,0***	15,1	***23,1***	...	15,23	***23,23***		***19,4***		***19,5***			***19,19***

Search area data — Reference window data

Fig. 21 Data re-allocation in phase 2.

Address	α+0	α+1	α+2	α+3	...	α+46	α+47	α+48	α+49	α+50	α+51	...	α+94	α+95	α+96	α+97	α+98	α+99	...	α+110	α+111
CRAM 0	0,0	8,0	0,1	8,1	...	0,23	8,23	8,0	16,0	8,1	16,1	...	8,23	16,23	***4,4***	12,4	***4,5***	12,5	...	***4,19***	12,19
CRAM 1	1,0	9,0	1,1	9,1	...	1,23	9,23	9,0	17,0	9,1	17,1	...	9,23	17,23	***5,4***	13,4	***5,5***	13,5	...	***5,19***	13,19
CRAM 2	2,0	10,0	2,1	10,1	...	2,23	10,23	10,0	18,0	10,1	18,1	...	10,23	18,23	***6,4***	14,4	***6,5***	14,5	...	***6,19***	14,19
CRAM 3	3,0	11,0	3,1	11,1	...	3,23	11,23	11,0	19,0	11,1	19,1	...	11,23	19,23	***7,4***	15,4	***7,5***	15,5	...	***7,19***	15,19
CRAM 4	4,0	12,0	4,1	12,1	...	4,23	12,23	12,0	20,0	12,1	20,1	...	12,23	20,23	***8,4***	16,4	***8,5***	16,5	...	***8,19***	16,19
CRAM 5	5,0	13,0	5,1	13,1	...	5,23	13,23	13,0	21,0	13,1	21,1	...	13,23	21,23	***9,4***	17,4	***9,5***	17,5	...	***9,19***	17,19
CRAM 6	6,0	14,0	6,1	14,1	...	6,23	14,23	14,0	22,0	14,1	22,1	...	14,23	22,23	***10,4***	18,4	***10,5***	18,5	...	***10,19***	18,19
CRAM 7	7,0	15,0	7,1	15,1	...	7,23	15,23	15,0	23,0	15,1	23,1	...	15,23	23,23	***11,4***	19,4	***11,5***	19,5	...	***11,19***	19,19

Search area data — Reference window data

Fig. 22 Data re-allocation in phase 3.

Then the upper 8 bits of each data are written to the destination addresses: $\alpha + 0$ to $\alpha + 92$ as shown in Fig. 20(a). In this re-allocation, the data are read from the CRAMs and written to the CRAMs using the simple addressing functions given by Eqs. (3) and (4) respectively. These addressing functions are implemented in AGUs.

$$Read\ address_{phase1\_sequence1} = t \tag{3}$$

$$Write\ address_{phase1\_sequence1} = 4t + \alpha \tag{4}$$

In the second sequence, the source addresses: 0 to 23 are read again and the lower 8 bits of the data are written to the destination addresses: $\alpha + 2$ to $\alpha + 94$ as shown in Fig. 20(b). The data are read from and written to the CRAMs using the addressing functions given by Eqs. (5) and (6) respectively. These addressing functions are implemented in AGUs.

$$Read\ address_{phase1\_sequence2} = t \tag{5}$$

$$Write\ address_{phase1\_sequence2} = 4t + (\alpha + 2) \tag{6}$$

Similarly, in the phase 2, the data in addresses: 12 to 43 (same as Fig. 16) are written to the addresses: $\alpha+1$ to $\alpha+111$ as shown in Fig. 21. For simplicity, the source addresses are not shown in Fig. 21. The data shown in bold-italic are the data that were copied during the phase 2. In the phase 3, the data in addresses: 44 to 51 (same as Fig. 16) are written to the addresses: $\alpha + 96$ to $\alpha + 110$ as shown in Fig. 22. The data shown in bold-italic are the data that were copied during phase 3.

4.2.3 SAD Calculation Using FE-GA

The FE-GA calculates the SADs between a reference block

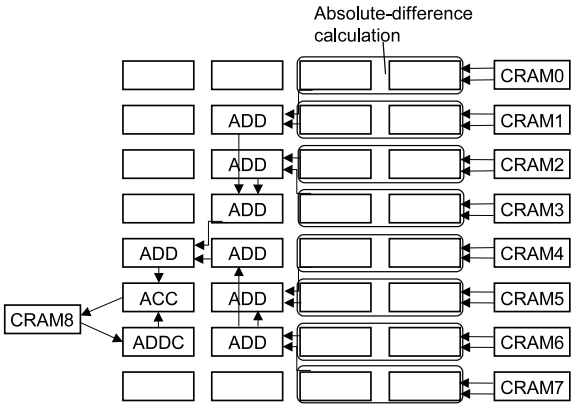


Fig. 23 SAD calculation using FE-GA.

and a candidate block. Figure 23 shows the mapping of SAD calculation to the FE-GA. Each CRAM contains the data of a reference and a candidate blocks. The absolute differences between pixels in the reference and candidate blocks are calculated using 8 pairs of ALUs. Absolute differences are added together using 7 adders. This value is accumulated for all the pixels in a candidate and a reference blocks. Then the SAD value is written to the CRAM 8. According to the specifications in Table 2, each reference blocks has 81 candidate blocks in the search area. Therefore, we calculate 81 SAD values per a reference block.

4.2.4 Search for Minimum SAD

The 81 SAD values are transferred to the local memory of the SH-4A using the DTU. The searching for the minimum SAD contain many control processing operations and it is not easy to implement it in the FE-GA. Therefore, we use the SH-4A for the minimum SAD search.

5. Evaluation

We evaluated the proposed method for block matching using the heterogeneous multi-core processor proposed in [3]. Figure 24 shows the processing time comparison between the proposed method and the method in [5] for one SH-4A and one FE-GA implementation. Note that, Fig. 24 shows the processing time of one corresponding pixel search. The data-transfer time is the time required to transfer the data from the DDR3-SDRAM to the CRAMs. The data-align time is the time required to re-align the data in the CRAMs. Note that the data-align time is necessary for the proposed method. The total clock cycles required for the block matching is only 31205 while 43958 are required in [5]. Therefore, the total processing time is reduced by 29%. The data-transfer time in method [5] is 29726 clock cycles. The sum of the data-transfer time and data-align time in the proposed method is only 16973 cycles. Therefore, the data-transfer time is reduced by more than 42% compared to that of [5]. The main reason for this is the acceleration of the data transfer using the DTU. The data-align overhead is very small

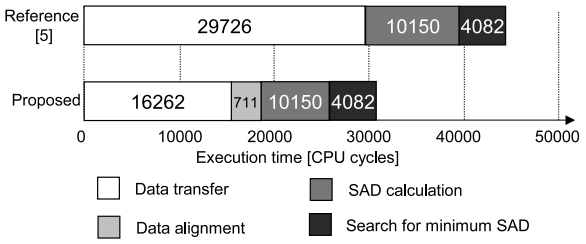


Fig. 24 Performance of the block-matching process.

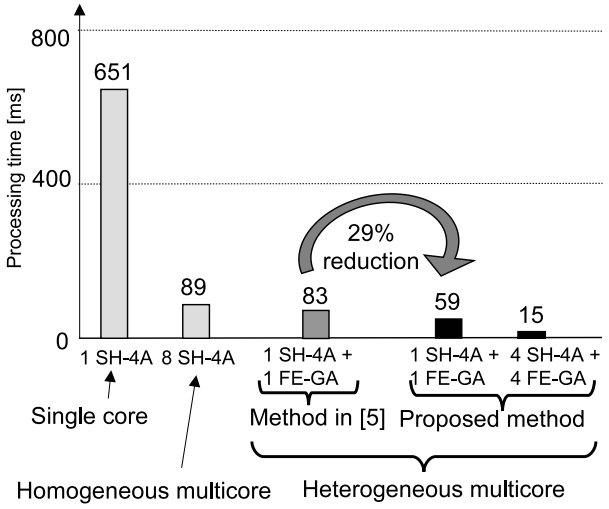


Fig. 25 Performance of heterogeneous vs. homogeneous processing.

compared to the data-transfer time due to the following reasons. The first is the fast address generation using the AGUs. The second is parallel access to 8 CRAMs.

1 SH-4A	:	a single SH-4A core performs SAD calculation and minimum SAD search.
8 SH-4A	:	8 SH-4A cores perform SAD calculation and minimum SAD search in parallel
1 SH-4A + 1 FE-GA	:	one SH-4A core and one FE-GA core are used for minimum SAD search and SAD calculation respectively.
4 SH-4A + 4 FE-GA	:	four pairs of a SH-4A core and an FE-GA core are used for parallel processing as explained in Sect. 4.2.

Figure 25 shows the processing time comparison of block matching for a VGA image using heterogeneous and homogeneous processing with and without the proposed method. As shown in Fig. 25, single SH-4A core implementation has the largest processing time. The processing time is reduced by 7.3 times using homogeneous multi-core processing with "8 SH-4A" cores. Performance similar to "8 SH-4A" implementation has been achieved in [5] using only two cores, one SH-4A and one FE-GA. Using the proposed method and using the same number of cores (one SH-4A and one FE-GA), we further reduced the processing time by 29%. In fig.25, we also compare 8 core homogeneous multi-core implemen-

tation with 8 core heterogeneous multi-core implementation (4 SF-4A and 4 FE-GA) using the proposed method. The processing time is reduced by 5.9 times in heterogeneous processing with the proposed method compared to homogeneous processing. This shows that, the heterogeneous processing with the proposed method gives significantly better performance compared to homogeneous processing.

Moreover, the processing time of "4 SH-4A+4 FE-GA" is one fourth of that of "1 SH-4A + 1 FE-GA". In other words, the processing time decreases linearly with the number of cores. When we use all 4 FE-GA accelerators in parallel in "4 SH-4A+4 FE-GA" implementation, we can process a VGA image at 15 ms. Such results shows that the heterogeneous processing with proposed method can be used in real-time image processing under the video frame-rate.

One reason for the processing time reduction in heterogeneous processing compared to homogeneous processing is the parallel processing in FE-GA. As shown in Fig. 23, eight absolute different calculations and 8 additions are done in parallel. Another reason is the AGU-based address generation in FE-GA. In FE-GA implementation, the address generation is done in parallel to the data processing in the ALU and MLT cells. However, in SH-4A implementation, addresses generation and data processing are done in serial using the same hardware. Due to these reasons the SAD calculation time that took 99% of the total processing time in [5] is reduced to 33% using the proposed method. The reason for the processing time reduction of 29% in the proposed method compared to the method in [5] is the proposed DTU-based data-transfer. Since the data-transfers among multiple cores is a major problem in heterogeneous processing, such a processing time reduction is a big achievement.

6. Conclusion

This paper presents a method to accelerate the data transfers exploiting data-transfer-units together with complex memory allocation. A flow of the method is that the data are initially transferred to the local memories using the data-transfer-module. Then, the data are aligned to the local memories so as to obtain the complex memory allocation reducing the data duplication. To verify the effectiveness of this method, we used block matching which is widely used in many image processing applications. It involves with a large amount of data and also requires complex memory access patterns. According to the results, the proposed method reduces the data-transfer time by more than 42% compared to that in conventional method. The main reason for this is the acceleration of the data transfer using the data-transfer-units. The another reason is that the data re-allocation overhead is very small since the address generation is done in AGUs and the data are aligned in parallel using multiple AGUs. Moreover, the processing time of the proposed method decreases linearly with the number of cores.

Acknowledgment

This work was supported by the New Energy and Industrial Technology Development Organization P05020, a joint project between Hitachi, Ltd., Renesas Electronics Corp., Waseda University, and Tokyo Institute of Technology. The design of this chip was supported by Yoichi Yuyama, Yoshikazu Kiyoshige, Yusuke Nitta, Masayuki Ito, Osamu Nishii, and Atsushi Hasegawa at Renesas Electronics Corp. and Tetsuya Yamada, Makoto Ishikawa, Masashi Takada, Takumi Nito, and Junichi Miyakoshi at Hitachi, Ltd. The design of the system was supported by Koichi Terada and Hiroyuki Mizuno at Hitachi, Ltd. The design of the parallelizing compiler was supported by Makoto Satoh at Hitachi, Ltd. and Yasutaka Wada, Akihiro Hayashi, Keiji Kimura, and Hironori Kasahara at Waseda University. The design of the software development envelopment was supported by Hideo Maejima at Tokyo Institute of Technology.

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] O. Takahashi, C. Adams, D. Ault, E. Behnen, O. Chiang, S.R. Cottier, P. Coulman, J. Culp, G. Gervais, M.S. Gray, Y. Itaka, and C.J. Johnson, "Migration of cell broadband engine from 65 nm SOI to 45 nm SOI," ISSCC Dig. Tech. Papers, pp.86–87, 2008.

[3] Y. Yuyama, M. Ito, Y. Kiyoshige, Y. Nitta, S. Matsui, O. Nishii, A. Hasegawa, M. Ishikawa, T. Yamada, J. Miyakoshi, K. Terada, T. Nojiri, M. Satoh, H. Mizuno, K. Uchiyama, Y. Wada, and K. Kimura, "A 45 nm 37.3GOPS/W heterogeneous multi-core SoC," ISSCC Dig., pp.100–101, 2010.

[4] K. Hosogi, S. Higashijima, T. Tashiro, A. Kawaguchi, and N. Nishioka, "A data transfer implementation on media processor MAPCA," IPSJ SIG Notes 2002, vol.9, pp.91–95, 2002. (in Japanese)

[5] 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, Jan. 2011.

[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, July 2005.

[7] S. Lee, M. Hariyama, and M. Kameyama, "An FPGA-oriented motionstereo processor with a simple interconnection network for parallel memory access," IEICE Trans. Inf. & Syst., vol.E83-D, no.12, pp.2122–2130, Dec. 2000.

[8] Z. Liu and X. Li, "XOR storage schemes for frequently used data patterns," J. Parallel Distrib. Comput., vol.25, pp.162–173, 1995.

[9] Y. Kobayashi, M. Hariyama, and M. Kameyama, "Optimal periodic memory allocation for image processing with multiple windows," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.17, no.3, pp.403–416, 2009.

[10] K. Uchiyama, F. Arakawa, H. Kasahara, T. Nojiri, H. Noda, Y. Tawara, A. Idehara, K. Iwata, and H. Shikano, "Heterogeneous Multicore Processor Technologies for Embedded Systems," Springer Science + Business Media, New York, 2012.

[11] H. Yamasaki, T. Kurafuji, M. Haraguchi, T. Nishijima, K. Murata, T.

Tanizaki, H. Noda, Y. Okuno, and K. Arimoto, "An energy-efficient massively parallel embedded processor core for real-time image processing SoC," Proc. COOLChips XIII, pp.395–410, 2010.

[12] K. Iwata, T. Irita, S. Mochizuki, H. Ueda, M. Ehama, M. Kimura, J. Takemura, K. Matsumoto, E. Yamamoto, T. Teranuma, K. Takakubo, and H. Watanabe, "A 342 mW mobile application processor with full-HD multi-standard video codec," ISSCC Dig. Tech. Papers, pp.158–159, 2009.

[13] M. Kimura, K. Iwata, S. Mochizuki, H. Ueda, M. Ehama, and H. Watanabe, "A full HD multi-standard video codec for mobile applications," IEEE Micro, vol.29, no.6, pp.18–27, 2009.

Yoshitaka Hiramatsu received a B.S. degree in information science and technology from Aichi Prefectural University, Japan, in 2002 and an M.E. degree in information engineering from Nagoya University, Japan, in 2004. From 2004, he has worked in the field of robot vision, image processing, image recognition, and very large-scale integration (VLSI) architecture design for video processing unit at Central Research Laboratory, Hitachi, Ltd., Tokyo, Japan (HCRL). He is a member of IEEE Computer Society.

Hasitha Muthumala Waidyasooriya received the B.E. degree in information engineering, and the M.S. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Miyagi, Japan, in 2006, 2008, and 2010, respectively. He is currently a Post-Doctoral Researcher with the Graduate School of Information Sciences, Tohoku University. His current research interests include heterogeneous multicore processor architectures and high performance computing.

Masanori Hariyama received the B.E. degree in electronic engineering, and the M.S. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Miyagi, Japan, in 1992, 1994, and 1997, respectively. He is currently an Associate Professor with the Graduate School of Information Sciences, Tohoku University. His current research interests include very large-scale integration (VLSI) computing for real-world application, such as robots, high-level design methodology for VLSIs, reconfigurable computing and high performance computing.

Tohru Nojiri received a B.E. degree in mathematical engineering from the University of Tokyo and a Ph.D. in information processing from Tokyo Institute of Technology. He is a senior researcher at Central Research Laboratory, Hitachi, Ltd. His research interests include embedded-system platforms, operating systems, and processor architectures. He is a member of IEEE Computer Society, the ACM, and the Information Processing Society of Japan.

Kunio Uchiyama received B.S. and M.S. degrees in information science from Tokyo Institute of Technology, Japan, in 1976 and 1978, respectively and a Ph.D. degree in advanced applied electronics from Tokyo Institute of Technology in 2001. Since 1978 he has been working for Central Research Laboratory, Hitachi, Ltd., Tokyo, Japan, on design automation, small-scale mainframes, cache memory, and microprocessors. From 1985 to 1986 he was a visiting researcher at the Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA. He also serves as a visiting professor at Waseda University. He got the Ichimura Aaward, R&D100, the Chief Officer's Award of the Japanese Science and Technology Agency, and the National Medal of Honor with Purple Ribbon in 1998, 1999, 2000, and 2004, respectively.

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 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. He is IEEE Fellow and IPSJ Fellow.