View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

東北大学機関リポジトリ
TOUR
Tohoku University Repository

Optimal Periodic Memory Allocation for Image Processing With Multiple Windows

著者	亀山　充隆
journal or publication title	IEEE Transactions on Very Large Scale Integration (VLSI) Systems
volume	17
number	3
page range	403-416
year	2009
URL	http://hdl.handle.net/10097/46854

doi: 10.1109/TVLSI.2008.2004547

Optimal Periodic Memory Allocation for Image Processing With Multiple Windows

Yasuhiro Kobayashi, *Member, IEEE*, Masanori Hariyama, *Member, IEEE*, and Michitaka Kameyama, *Fellow, IEEE*

***Abstract*—One major issue in designing image processors is to design a memory system that supports parallel access with a simple interconnection network. This paper presents an efficient memory allocation to minimize the number of memory modules and processing elements with a parallel access capability when multiple windows with arbitrary shapes are specified. This paper also presents an efficient search method based on regularity of window-type image processing. We give some practical examples including a stereo-matching processor for acquiring 3-D information, and an optical-flow processor for motion estimation. These examples show that the numbers of memory modules are reduced to 2.7% and 10%, respectively, in comparison with a basic approach. It is also shown that the search time is less than 1 ms for practical image sizes and window sizes.**

***Index Terms*—Image processing, memory design, optimization, parallel processors.**

I. Introduction

HIGHLY-PARALLEL image processors require a complex interconnection network between memory modules and processing elements (PEs) for parallel memory access. One typical image processor consists of memory modules, PEs, interconnection network between memory modules and PEs, and an inter-PE network. The complexity of the interconnection network between the memory modules and PEs increases with the number of memory modules, and it causes significant overhead in delay and power in deep-submicrometer and more advanced technologies since the delay and the power of interconnection units are more dominant than those of logic units. The interconnection problem is also serious in image processors using field-programmable gate arrays (FPGAs) that have large interconnection delays because of complex programmable switch blocks. To solve the problem, we introduce an architecture where a memory module is connected to a single processing element. This architecture model can be though of as a simplified form of recent image processors based on single instruction multiple data (SIMD) architecture [1], [2]. In the architecture, the total hardware amount linearly increases with the number of memory modules. Memory allocation has a great impact on the number of memory modules. Therefore, this paper presents memory allocation to minimize the number of memory modules with a parallel access capability.

Manuscript received September 20, 2007; revised February 06, 2008. First published January 20, 2009; current version published February 19, 2009.

Y. Kobayashi is with Oyama National College of Technology, Oyama 323-0806, Japan (e-mail: y-kobayashi@oyama-ct.ac.jp).

M. Hariyama and M. Kameyama are with Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan,.

Digital Object Identifier 10.1109/TVLSI.2008.2004547

This paper targets window-type image processing. The window-type image processing is widely used in practical applications. Its examples include filtering, template matching and morphology. An application usually requires several types of windows. The memory allocation must support parallel access for all types of windows that is given as a specification.

There are a number of research works which have addressed memory allocation [3]–[11]. They are classified into two groups: [3]–[8] and [9]–[11]. The first group handles array-variable clustering whereby one or more array variables are stored in the same memory module based on cost and performance considerations. Reference [3] minimizes the area. Reference [4] considers memory hierarchy to handle the trade-off between performance and cost. Reference [5] minimizes page misses in a bank while respecting data dependences. Reference [6] minimizes energy and/or area based on instruction-level parallelism (ILP)-based memory allocation. Reference [8] also minimizes energy based on scheduling as well as memory allocation. In the array-variable clustering, data in an array are basically allocated to the same memory module/bank. Therefore, it is difficult to exploit parallelism between data in an array.

The second group divides an array into several sub-arrays, and allocates sub-arrays to memory modules. This type of memory allocation allows to exploit parallelism between data in an array by distributing the concurrently-accessed sub-arrays between different memory modules. This paper targets window-type image processing using multiple windows. In practical cases, an entire image processing is made up of several sub-tasks such as filtering, template matching, segmentation and so on. Window-type image processing is frequently encountered in filtering and template matching. Filtering is used as preprocessing for noise removal, smoothing and edge detection and so on which require different windows. In each processing such as edge detection, several windows may be used. For example, the Sobel edge detection, the Marr–Hildreth edge detection [12], and the K-forms edge detection [13] require respectively, two, two, and eight different windows. Moreover, mathematical morphology [14]–[18] is one of the window-type image processing. Mathematical morphology [14]–[18] is frequently used for filtering and geometric analysis by structuring elements. In many applications, it is advantageous to use different structuring elements with various sizes and shapes. Structuring elements correspond to windows. In image processing, exploiting pixel-level parallelism plays an essential role to speed up. Hence, this paper focuses on the second group. This type of memory allocation takes into account regularity of target processing to reduce the larger search

1063-8210/$25.00 © 2009 IEEE

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON BETWEEN THIS WORK AND PREVIOUS WORKS IN TERMS OF PROBLEM DEFINITION

	[10]	[9]	[11]	This work
Resolution	Single	Single	Multi	Multi
Window shape and number	Single square shape	Single arbitrary shape	Single square shape	Multiple arbitrary shapes
Objective function	Number of memory modules	Power consumed by address bus	Number of memory modules	Number of memory modules
Periodicity (directions and their periods)	Horizontal and vertical, same periods	Horizontal and vertical, different periods	Horizontal and vertical, same periods	Arbitrary two directions, different periods
Optimality	Optimal	Heuristic	Optimal	Optimal

space than the array-variable clustering. [10] is originally for motion stereo, and can be applied to a window-type image processing with a single square window at a single resolution. Its objective is to minimize the number of memory modules. The whole image is equally divided into square regions of the same size as the window. The pixels of a square region are allocated to different memory modules. The memory allocation in a square region is repeated horizontally and vertically throughout the image. As a result, the pixels in a window are distributed among different memory modules. Reference [9] addresses the problem of system power reduction through transition count minimization on the memory address bus when arrays in behavioral specifications are accessed from memory modules. It targets for memory-intensive applications such as digital signal processing, and image processing that exhibit regular access patterns. The authors exploit regularity and spatial locality in the memory. When considering an image as a 2-D array, it is applicable for image processing. The 2-D array is equally divided into square regions of the same size called "tiles". The memory allocation for a tile is repeated horizontally and vertically. The objective function is to minimize the power consumed by address bus. This method is partially applicable to the minimization of the number of memory modules. For the minimization of the number of memory modules, it provides the same result as [10], because it exploits horizontal and vertical regularity. [11] is originally for stereo matching with a hierarchical matching approach to reduce the computational amount. However, it handles a single square window unlike our method. Table I summarizes the features of the previous works and this works. In terms of image resolution, only this work and [11] handle multi-resolution. In terms of window shape and the number, only this work handles multiple and arbitrary windows. In terms of periodicity, only this work handles arbitrary directions with arbitrary periods. This allows us to reduce the number of memory modules compared to the previous works. Although search space is expanded by considering arbitrary directions with arbitrary periods, this work guarantees to find optimal solution.

Given multiple windows, the most simple way of finding the memory allocation for parallel access is to apply [10] (or [9]) to the minimum rectangle window that includes all the given windows. The concept behind this method is to approximate the multiple windows by a single rectangle window. However, this method can require the large number of memory modules when the approximation is not good. In this paper, for further reduction of the required number of memory modules, a whole image is equally divided into parallelograms, and the memory allocation for a parallelogram is repeated along the sides of the parallelogram. A parallelogram is formed by a pair of vectors with different lengths and different directions. Since a parallelogram is a generalized type of a square, its use results in further reduction of the required memory modules. Its disadvantage over the rectangle-window-based approach is the larger search space. To solve this problem, this paper proposes an efficient search method based on the regularity of window-type processing. To reduce search space, search for several vector pairs can be replaced with search for a vector pair called "equivalent vector pair" which provides the same memory allocation as them.

This paper is organized as follows. In Section II, we formulate the memory allocation problem. In Section III, we show an efficient search method based on the regularity of a periodic memory allocation. In Section IV, we give some practical examples including a stereo-matching processor for acquiring 3-D information, and an optical-flow processor for motion estimation. These examples show that the numbers of memory modules are reduced to 2.7% and 10%, respectively, in comparison with the conventional method [10]. They also show that the search time is less than 1 ms on a PC (Pentium4 at 2 GHz) for practical image sizes and window sizes. In Section V, we discuss the extension of the proposed method to more practical problems. In Section VI, we state our conclusions.

II. PROBLEM FORMULATION

A. Target Processing

We consider a window-type image processing. Let us begin with image processing using a single window. In this type of processing, the output/intermediate output depends on a small neighborhood of an input image, where the neighborhood size is fixed and given as a window. Algorithms of this type frequently appear in practical situations: spatial filter, morphology, and image matching, and so on. Moreover, they usually have the high degree of parallelism, and are suited for VLSI implementations.

We use window-serial-and-pixel-parallel scheduling as shown in Fig. 1. In this scheduling, operations are performed in parallel with pixels in a window, whereas operations are performed in a serial manner for windows. Fig. 1(a) shows the location of the window at each step. The thick line denotes a window. Fig. 1(b) shows the scheduled data-flow graph (SDFG) corresponding to Fig. 1(a). A node in the SDFG denotes an operation. The labels $\mathbf{A}$ and $\mathbf{B}$ on operations denote the operation types of the nodes. There is an edge between nodes when the output of one node is used as an input of the other node. At Step 1 in Fig. 1, pixels: (0,0), (0,1), (0,2) and (1,1) are used as inputs of operations of type $\mathbf{A}$. These pixels must be accessed in parallel since the $\mathbf{A}$-type operations are performed in parallel. Their results are used as inputs of the $\mathbf{B}$-type operation. As the location of the window changes, the input pixels change.

An image-processing system usually requires various windows for filtering, edge detection, morphology, and so on.

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

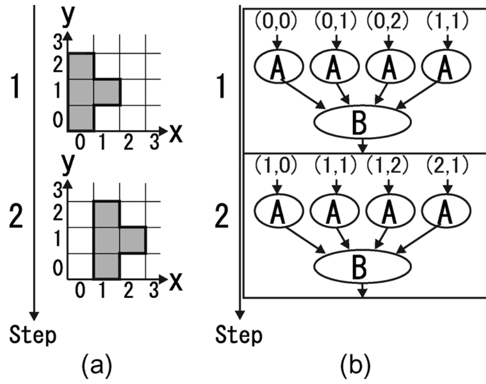


Fig. 1. DFG of a window-serial-and-pixel-parallel schedule for a window-type processing with a single window denoted by a set of gray pixels. (a) Image plane. (b) SDFG.

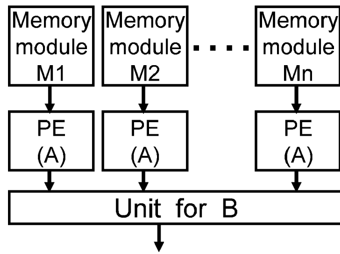


Fig. 2. Target architecture.

Hence, the image-processing system requires a memory allocation that enables parallel access for such various windows.

B. Target Architecture

Fig. 2 shows our target architecture. A single PE is connected to each memory module. Single-port memories are used as memory modules. Multi-port memories are also one efficient method for parallel access. Although an N-port memory provides more flexibility than N single-port memories, the use of a multi-port memory imposes a larger hardware amount because of additional bit-lines and decoders. Multi-port memories are efficient especially in the case when access patterns are not predetermined. Window-type processing has a regular access pattern. Hence, it is possible that pixels in a window are distributed among multiple single-port memory modules by appropriate memory allocation. As a result, use of single-port memories is suitable for window-type processing because of the smaller area. All pixels of an input image are distributed among the memory modules. The PEs perform operations of type $\mathbf{A}$ according to the window-serial-and-pixel-parallel scheduling shown in Fig. 1(b). Their outputs are used as inputs of the unit for type-$\mathbf{B}$ operations. In order to extend the architecture model, you can add inputs to PEs as required. In the target architecture, the hardware amount is determined by a memory allocation task since the number of PEs is determined by the number of memory modules. Therefore, a memory allocation plays an essential role in minimizing hardware the amount of the target architecture.

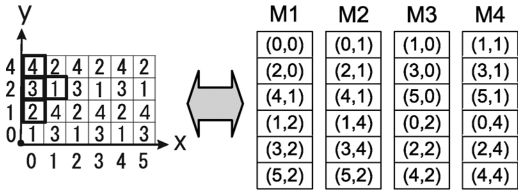
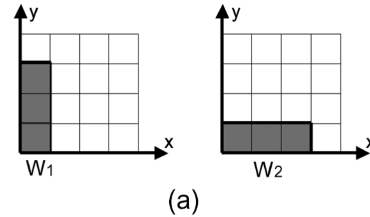


Fig. 3. Memory allocation.



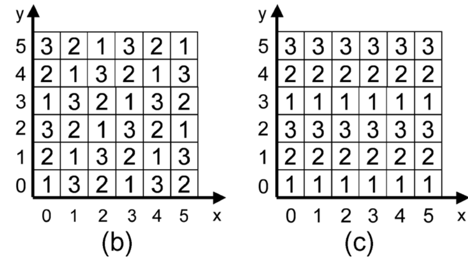


Fig. 4. Memory allocation for two windows. (a) Windows. (b) Allocation1. (c) Allocation2.

C. Memory Allocation for Image Processing

Memory allocation is a task that assigns pixels to memory modules. Fig. 3 shows an example of memory allocation for the window shown in Fig. 1(a). The label on each pixel denotes the memory module to which the pixel is assigned. For example, pixels: (0,0), (0,1), (1,0) and (1,1) are assigned to memory modules: M1, M2, M3 and M4, respectively.

To meet the timing constraint of the window-serial-and-pixel-parallel scheduling, all the pixels in a window must be accessed in parallel for all possible locations of the window. In other words, all the pixels in a window must be distributed among different memory modules for all possible locations of the window. Readers can examine that the memory allocation shown in Fig. 3 enables the parallel memory access for any location of the window.

For a memory allocation for image processing with multiple windows, all the pixels in each window must be accessed in parallel for all possible locations of the window. Fig. 4 shows an example of memory allocation for two windows. Fig. 4(a) shows windows: W_1 and W_2. Fig. 4(b) shows the memory allocation result that allows parallel access for the windows. All pixels in each window are distributed among different memory modules for all possible locations of the window. Fig. 4(c) shows the memory allocation that is not capable of parallel access. All pixels in W_1 are distributed among different memory modules for all possible locations of the windows. This memory allocation enables the parallel memory access for W_1. On the other hand, all pixels in W_2 are allocated to a same memory module. This memory allocation does not enable the parallel memory access for W_2.

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

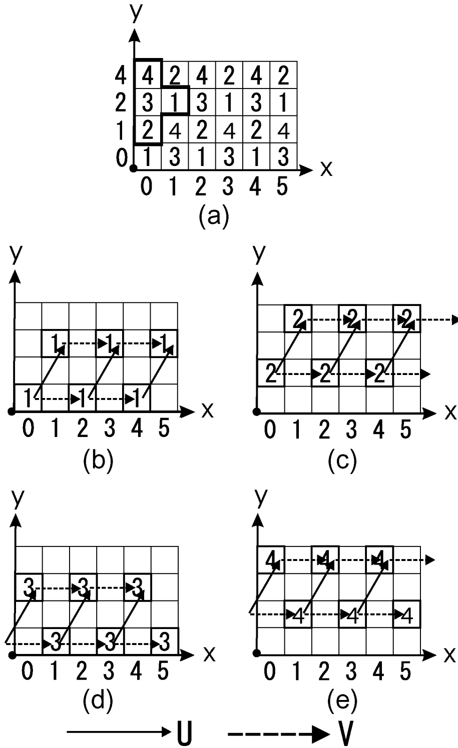


Fig. 5. Example of a periodic memory allocation.

D. Periodic Memory Allocation

From a practical point, a memory allocation should have a simple addressing function to determine which memory module stores each pixel. If the addressing function is complex, the area and delay of an addressing circuit become large. In the worst case, a lookup table for all the pixels is required for addressing. The periodic memory allocation has a simple addressing function because of its regularity. Let K be the number of memory modules. Let N_i be the number of pixels that are allocated to the memory module M_i for $1 \leq i \leq K$. Let (x_i^j, y_i^j) be the coordinates of a pixel that is allocated to the memory module M_i for $1 \leq j \leq N_i$. Then, we define a periodic memory allocation as a memory allocation where (x_i^j, y_i^j) is expressed as

$$\begin{bmatrix} x_i^j \\ y_i^j \end{bmatrix} = s_i^j \begin{bmatrix} U_x \\ U_y \end{bmatrix} + t_i^j \begin{bmatrix} V_x \\ V_y \end{bmatrix} + \begin{bmatrix} x_i^0 \\ y_i^0 \end{bmatrix} \quad (1)$$

where $\boldsymbol{U} = [U_x\ U_y]^T$ and $\boldsymbol{V} = [V_x\ V_y]^T$ are vectors to represent periods (called period vectors); the variables s_i^j and t_i^j are integers; the coordinates (x_i^0, y_i^0) are coordinates of the reference pixel allocated to M_i. Note that you can select an arbitrary pixel as the reference pixels from the pixels allocated to the same memory module.

Fig. 5(a) shows an example of a periodic memory allocation for the SDFG shown in Fig. 1. The label on a pixel denotes which memory module stores each pixel. For example, the pixels: (0,0), (0,1), (1,0) and (1,1) are allocated to M_1, M_2, M_3, and M_4, respectively. From Fig. 5(b), the coordinates of the pixels allocated to M_1 are given by

$$\begin{bmatrix} x_1^j \\ y_1^j \end{bmatrix} = s_1^j \begin{bmatrix} 1 \\ 2 \end{bmatrix} + t_1^j \begin{bmatrix} 2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (2)$$

where the coordinates of the reference pixel and the period vectors are $[0\ 0]^T$, $\boldsymbol{U} = [1\ 2]^T$, and $\boldsymbol{V} = [2\ 0]^T$, respectively. The pixels with label 1 in Fig. 5(b) are given by $(s_1^j, t_1^j) = (0, 0)$, (1,0), (2,0), (0,1), (1,1) and (2,1), respectively. Figs. 5(c)–(e) show the pixels allocated to M_2, M_3, and M_4, respectively. These figures show that the same period vectors as M_1 are used for M_2, M_3, and M_4. The memory allocation shown in Fig. 5 satisfies (1), that is, the definition of a periodic memory allocation.

As mentioned at the beginning of this section, the advantage of the periodic memory allocation is its simple addressing function. For the periodic memory allocation shown in Fig. 5(a), the addressing function $f(x, y)$, which allocates pixel (x, y) to memory module $M_{f(x,y)}$, is given by

$$f(x, y) = (2x + y) \bmod 4 + 1. \quad (3)$$

E. Optimal Memory Allocation

Let us minimize the number of memory modules when windows: $W_1, W_2 \ldots$, and W_n are specified. The optimal memory allocation is defined as the memory allocation that satisfies the following conditions.

- C1) For any location of each window, all pixels in the window can be retrieved in parallel. In other words, all pixels in each window are allocated to different memory modules.
- C2) Each pixel is allocated to a single memory module. This condition ensures that the total memory capacity is minimized.
- C3) The number of memory modules is minimized. This condition ensures that the hardware amount is minimized in the target architecture as mentioned in Section II-B.
- C4) The memory allocation is a periodic one. This condition ensures that the hardware for the addressing function is small.

Fig. 4(b) gives an example of the optimal memory allocation for multiple windows shown in Fig. 4(a). The condition C1 is satisfied for W_1 and W_2. The condition C2 is satisfied since each pixel has a single label. The condition C3 is satisfied since the number of memory modules is 3 and is equal to the minimum number of memory modules required for parallel access, i.e., the number of pixels in each window. The condition C4 is satisfied since the memory allocation is obtained from (1) with $\boldsymbol{U} = [3, 0]$ and $\boldsymbol{V} = [1, 1]$. Fig. 5(a) also gives an example of the optimal memory allocation for a single window shown in Fig. 1(a). Readers can examine it in the similar manner with Fig. 4(b).

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

III. Search Method

A. Estimation of the Number of Memory Modules

Given the period vectors $\boldsymbol{U}$ and $\boldsymbol{V}$, the number of memory modules is estimated by the area of the parallelogram made by the period vectors $\boldsymbol{U}$ and $\boldsymbol{V}$. The area S of the parallelogram is given by

$$S = |U_x \times V_y - U_y \times V_x|. \tag{4}$$

For the example shown in Fig. 5

$$S = |1 \times 0 - 2 \times 2| = 4 \tag{5}$$

and S is exactly the same as the number of memory modules. This is because the memory allocation for a whole image is given by repeating the memory allocation for the parallelogram, and the parallelogram must be filled with pixels allocated to different memory modules. From these observations, finding the optimal memory allocation is reduced to finding period vectors that make the minimum parallelogram still satisfying the parallel access condition.

B. Basic Search Algorithm

We suppose that windows: $W_1, W_2, \ldots, W_n$ are specified. Period vectors $\boldsymbol{U}$ and $\boldsymbol{V}$ are treated as a pair (called a vector pair $(\boldsymbol{U}, \boldsymbol{V})$).

We explain variables used in this algorithm. For $i = 1, 2, \ldots, n$, let I_i and H_i be the width and height of the minimum bounding rectangle of W_i, respectively. Let I and H be the width and height of the rectangular window that includes all the windows, respectively. The values I and H are given by

$$\begin{aligned} I &= \max\{I_1, I_2, \ldots, I_n\} \\ H &= \max\{H_1, H_2, \ldots, H_n\}. \end{aligned} \tag{6}$$

For example, let us consider two windows: W_1 and W_2 shown in Fig. 6(a) and (b), respectively. Fig. 6(c) and (d) show the minimum bounding rectangles of W_1 and W_2, respectively. From (6), $I = \max\{I_1, I_2\} = \max\{2, 3\} = 3$, $H = \max\{H_1, H_2\} = \max\{3, 2\} = 3$. As a result, we obtain the minimum bounding rectangle of W_1 and W_2 as shown in Fig. 6(e). Let S be the area of the parallelogram made by a vector pair $(\boldsymbol{U}, \boldsymbol{V})$, that is, the number of memory modules of the memory allocation obtained by the vector pair. Let $S_{\min}$ be the current minimum number of memory modules. Let G be the set of vector pairs to be checked.

Fig. 7 shows the outline of the search algorithm. Lines 1–3 initialize variables. The initial value of $S_{\min}$ is determined by the rectangular memory allocation [10]. As mentioned in the previous paragraph, all the windows are approximated by the rectangular window of size $I \times H$. It is mathematically guaranteed that the rectangular memory allocation provides the memory allocation with the minimum number of memory modules and capability of completely-parallel access. To obtain an allocation for a whole image, the rectangular memory allocation regularly repeats the allocation for the rectangular region. For example, Fig. 8 shows the result of the rectangular memory allocation for the rectangular window shown in Fig. 6(e). Hence, the initial value of $S_{\min}$ is defined by the number of memory modules for the rectangular window of size $I \times H$, that is

$$S_{\min} = I \times H. \tag{7}$$

For the example of Fig. 6

$$S_{\min} = 3 \times 3 = 9. \tag{8}$$

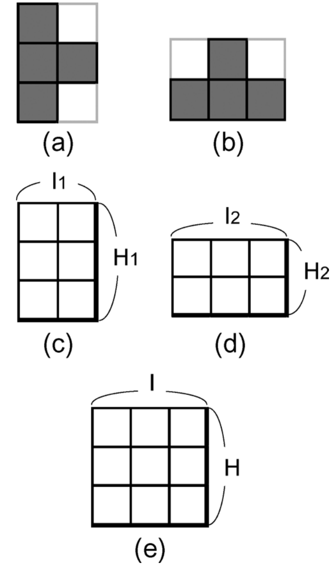


Fig. 6. Rectangular window. (a) W_1; (b) W_2; (c) bounding rectangle for W_1 ($I_1 = 2$, $H_1 = 3$); (d) bounding rectangle for W_2 ($I_2 = 3$, $H_2 = 2$); (e) bounding rectangle for W_1 and W_2 ($I = 3$, $H = 3$).

```
1:  Smin = I·H;
2:  Umin = (I,0); Vmin = (0, H);
3:  Initializing G

4:  for each vector-pair (U,V) ∈G do
5:    G = G - {(U,V)};
6:    S = MODULE_NUMBER(U,V);

7:    if CHECK_PARALLEL_ACCESS(U,V) then
8:      if S < Smin then
9:        Smin = S;
10:       Umin = U; Vmin = V;
11:     endif
12:   endif
13: endfor
```

Fig. 7. Basic algorithm.

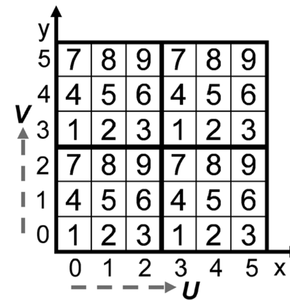


Fig. 8. Result of rectangular memory allocation.

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

The initial values of $\boldsymbol{U}_{\min}$ and $\boldsymbol{V}_{\min}$ are also determined by the rectangular memory allocation. As shown in Fig. 8, the rectangular memory allocation is a periodic one. Hence, the initial values of $\boldsymbol{U}_{\min}$ and $\boldsymbol{V}_{\min}$ are given by period vectors of the rectangular memory allocation, that is

$$\boldsymbol{U}_{\min} = (I, 0), \quad \boldsymbol{V}_{\min} = (0, H). \tag{9}$$

For the example shown in Fig. 6

$$\boldsymbol{U}_{\min} = (3, 0), \quad \boldsymbol{V}_{\min} = (0, 3). \tag{10}$$

The initial value of G is expressed as

$$G = \{(\boldsymbol{U}, \boldsymbol{V}) | \boldsymbol{U} = (U_x, U_y), \boldsymbol{V} = (V_x, V_y), 0 \le U_x < X_{\max}, \\ 0 \le U_y < Y_{\max}, 0 \le V_x < X_{\max}, 0 \le V_y < Y_{\max}\} \tag{11}$$

where $X_{\max}$ and $Y_{\max}$ are the maximum values of X and Y coordinates, respectively. Let $(\boldsymbol{U}_{\min}, \boldsymbol{V}_{\min})$ be the current optimal vector pair.

The "**for**" loop in Fig. 7 finds a vector pair which makes the optimal memory allocation from all vector pairs. The function $MODULE_NUMBER(\boldsymbol{U}, \boldsymbol{V})$ returns the number of memory modules obtained by (4). The function $CHECK_PARALLEL_ACCESS(\boldsymbol{U}, \boldsymbol{V})$ returns true if the memory allocation obtained by $(\boldsymbol{U}, \boldsymbol{V})$ satisfies the condition C1, that is, the parallel access condition. In each iteration, the variables $(\boldsymbol{U}_{\min}, \boldsymbol{V}_{\min})$ and $S_{\min}$ maintain the optimal vector pair and the area of parallelogram made by $(\boldsymbol{U}_{\min}, \boldsymbol{V}_{\min})$, respectively. During each iteration, $(\boldsymbol{U}_{\min}, \boldsymbol{V}_{\min})$ and $S_{\min}$ are updated if the number of memory modules, which is obtained by $MODULE_NUMBER(\boldsymbol{U}, \boldsymbol{V})$, is less than $S_{\min}$.

We explain a straightforward way for $CHECK_PARALLEL_ACCESS(\boldsymbol{U}, \boldsymbol{V})$. First, the memory allocation for a whole image is made by a vector pair $(\boldsymbol{U}, \boldsymbol{V})$. When $(\boldsymbol{U}, \boldsymbol{V})$ is given, pixels stored in each memory module are determined by (1) as described in Section II-D. Next, for each of the windows: $W_1, W_2, \ldots, W_n$, we check whether all the pixels in the window are allocated to different memory modules for all the locations of the window. This checking is time-consuming in a straightforward way. An efficient algorithm based on the periodicity of the periodic memory allocation is presented in Section III-C4.

C. Improving Search Efficiency

1) Equivalent Vector Pair to Reduce the Search Space: Search for several vector pairs can be replaced with search for a vector pair called "*equivalent vector pair*". Let us consider a vector pair $(\boldsymbol{A}, \boldsymbol{B})$ for vectors $\boldsymbol{A} = [A_x, A_y]^T$ and $\boldsymbol{B} = [B_x, B_y]^T$, where the elements of $\boldsymbol{A}$ and $\boldsymbol{B}$ are integer. We call the vector pair $(\boldsymbol{A}, \boldsymbol{B})$ the equivalent vector pair of $(\boldsymbol{U}, \boldsymbol{V})$ if $(\boldsymbol{A}, \boldsymbol{B})$ satisfies the following conditions.

D1) The linear combination of $\boldsymbol{U}$ and $\boldsymbol{V}$ with integer coefficients can be given by the linear combination of $\boldsymbol{A}$ and $\boldsymbol{B}$ with integer coefficients, or vice versa. This can be expressed as

$$k\begin{bmatrix} U_x \\ U_y \end{bmatrix} + l\begin{bmatrix} V_x \\ V_y \end{bmatrix} = m\begin{bmatrix} A_x \\ A_y \end{bmatrix} + n\begin{bmatrix} B_x \\ B_y \end{bmatrix} \tag{12}$$

where k, l, m, and n are integer scalars.

D2) The area of the parallelogram made by $(\boldsymbol{U}, \boldsymbol{V})$ is equal to the area of the parallelogram made by $(\boldsymbol{A}, \boldsymbol{B})$.

D3) The vector $\boldsymbol{A}$ is parallel to the x-axis. In other words, $A_y = 0$.

The conditions D1 and D2 denote that a memory allocation obtained by a vector pair $(\boldsymbol{U}, \boldsymbol{V})$ is equal to a memory allocation obtained by a vector pair $(\boldsymbol{A}, \boldsymbol{B})$. There are some vector pairs which satisfy the condition D1 and D2. The condition D3 is used to uniquely select the vector pair from them.

```
1:  B1x = Ux; B1y = Uy;
2:  B2x = Vx; B2y = Vy;

3:  while B1y ≠ B2y do
4:    if B1y < B2y then
5:      B2x = B2x – B1x;
6:      B2y = B2y – B1y;
7:    else
8:      B1x = B1x – B2x;
9:      B1y = B1y – B2y;
10:   endif
11: endwhile
12: By = B1y;

13: Ax = |Ux × Vy – Uy × Vx| / By;

14: Bx = B1x;
15: if Bx ≧ 0 then
16:   Bx = Bx mod Ax;
17: else
18:   while Bx < 0 do
19:     Bx = Bx + Ax;
20:   endwhile
21: endif
```

Fig. 9. Deriving an equivalent vector pair.

Fig. 9 shows a method of deriving an equivalent vector pair $\boldsymbol{A} = (A_x, 0)$ and $\boldsymbol{B} = (B_x, B_y)$. Lines 1 and 2 initialize variables. Lines 3–12 basically calculate

$$B_y = \mathrm{GCM}(U_y, V_y)$$

using the Euclidean algorithm, where $\mathrm{GCM}(U_y, V_y)$ is the greatest common measure of U_y and V_y. Line 12 calculates A_x. Since there are several choices of B_x, lines 14 to 21 uniquely select B_x with the minimum positive x-coordinate. For example, Fig. 10(a) shows an equivalent vector pair $(\boldsymbol{A}, \boldsymbol{B})$ of vector pairs $(\boldsymbol{U}, \boldsymbol{V})$ shown in Figs. 10(b)–(d). Gray pixels denote linear combinations of $\boldsymbol{U}$ and $\boldsymbol{V}$ with integer coefficients, and they are stored in the same memory modules. Let us prove that the vector pair shown in Fig. 10(a) is the equivalent vector pair of the vector pair shown in Fig. 10(b). The two vector pairs $(\boldsymbol{U}, \boldsymbol{V})$ and $(\boldsymbol{A}, \boldsymbol{B})$ satisfy the condition D1 since gray pixels in Fig. 10(b) conform to those in Fig. 10(a). The area of parallelogram obtained by the vector pair $(\boldsymbol{U}, \boldsymbol{V})$ is given by

$$|3 \times 9 - 6 \times 2| = 15. \tag{13}$$

The area of parallelogram obtained by the vector pair $(\boldsymbol{A}, \boldsymbol{B})$ is given by

$$|5 \times 3 - 0 \times 4| = 15. \tag{14}$$

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

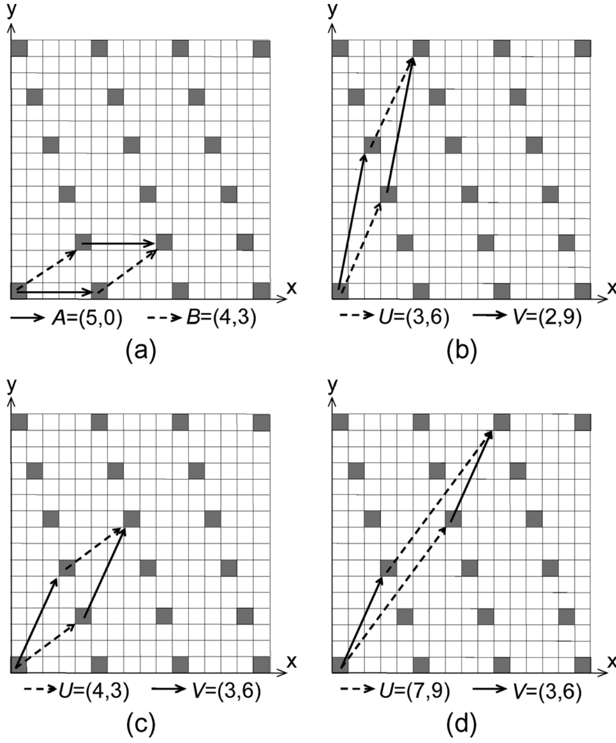


Fig. 10. Equivalent vector pair.

Therefore, the two vector pairs $(\boldsymbol{U}, \boldsymbol{V})$ and $(\boldsymbol{A}, \boldsymbol{B})$ satisfy the condition D2. The vector pair $(\boldsymbol{A}, \boldsymbol{B})$ satisfies the condition D3 since the y-coordinate of $\boldsymbol{A}$ is 0. Similarly, readers can prove that the vector pair shown in Fig. 10(a) is the equivalent vector pair of $(\boldsymbol{U}, \boldsymbol{V})$ shown in Fig. 10(c) and (d). By using the equivalent vector pair, search for three vector pairs can be replaced with the search for a single equivalent vector pair. As a result, we can reduce a search space for vector pairs.

We omit proving that an equivalent vector pair $(\boldsymbol{A}, \boldsymbol{B})$ for an arbitrary vector pair $(\boldsymbol{U}, \boldsymbol{V})$ exists because the length of this paper is limited.

2) Reduction of Computational Complexity Based on Constraint of the Area of Parallelogram: We can reduce a search space by using a constraint of the area of a parallelogram obtained by a vector pair. For an equivalent vector pair $(\boldsymbol{A}, \boldsymbol{B})$, the area S of the parallelogram made by $(\boldsymbol{A}, \boldsymbol{B})$ is given by

$$S = |A_x \times B_y - A_y \times B_x| \tag{15}$$

from (4). Note that S also means the number of memory modules for the memory allocation obtained by $(\boldsymbol{A}, \boldsymbol{B})$. For a window with K pixels, at least K memory modules must be used to enable parallel access. This condition is expressed as

$$K \leq S. \tag{16}$$

We search for a vector pair with the smaller area than $S_{\min}$ to find the optimal memory allocation as shown in Lines 8–10, Fig. 7. From (4), this condition is expressed as

$$S < S_{\min}. \tag{17}$$

```
1:  Smin = I·H;
2:  Amin = (I,0); Bmin = (0, H);
3:  Initializing G

4:  for each vector-pair (A,B)∈G do
5:    G = G - {(A,B)};
6:    S = MODULE_NUMBER(A,B);

7:    if K ≦ S < Smin then
8:      if CHECK_PARALLEL_ACCESS(A,B) then
9:          Smin = S;
10:         Amin = A; Bmin = B;
11:     endif
12:   endif
13: endfor
```

Fig. 11. Improved algorithm.

Hence, when $S_{\min}$ and K are given, the search space of $(\boldsymbol{A}, \boldsymbol{B})$ is limited to

$$K \leq S < S_{\min}. \tag{18}$$

3) Improved Search Algorithm: Based on the methods mentioned in Sections III-C1 and III-C2, we obtain an improved algorithm shown in Fig. 11. The differences between the improved algorithm and the basic one are as follows. First, we use an equivalent vector pair $(\boldsymbol{A}, \boldsymbol{B})$ in place of a vector pair $(\boldsymbol{U}, \boldsymbol{V})$. Second, the number of elements of G is reduced. When initializing G at the third Line, Fig. 11, G is set to

$$G = \{(\boldsymbol{A}, \boldsymbol{B}) | \boldsymbol{A} = (A_x, A_y), \boldsymbol{B} = (B_x, B_y), 0 \leq A_x < X_{\max}, \\ A_y = 0, 0 \leq B_x < X_{\max}, 0 \leq B_y < Y_{\max}\} \tag{19}$$

from the condition D3. Hence, the number of elements of G is $X_{\max}^2 Y_{\max}$ in the improved algorithm from (19). On the other hand, the number of elements of G is $X_{\max}^2 Y_{\max}^2$ from (11) in the basic algorithm. Third, the number of vector pairs to perform $CHECK_PARALLEL_ACCESS$ is reduced as shown in Line 7 of Fig. 11.

4) Reduction of Computational Complexity of Parallel Access Check: Let us improve the function $CHECK_PARALLEL_ACCESS$ to reduce the computational amount. Let us consider an arbitrary pixel P. We define "*parallel access pattern*" of P as the set of pixels that are possibly accessed in parallel together with P. By using the parallel access pattern, the condition C1 is rewritten as "**The pixels in the parallel access pattern of *P* are allocated to different memory modules from the memory module of *P***".

Let us make the parallel access pattern of P when a window is specified. The pixel P can take an arbitrary location in the window. When the window shown in Fig. 6(a) is specified, the possible locations of P are shown in Fig. 12(a)–(d), respectively. Gray pixels in each figure are accessed in parallel together with P. When the coordinates of P are (x, y), the gray pixels shown in Fig. 12(a)–(d) are expressed as (20)–(23), respectively

$$\{(x,y),(x,y+1),(x+1,y+1),(x,y+2)\} \tag{20}$$

$$\{(x,y),(x+1,y),(x,y+1),(x,y-1)\} \tag{21}$$

$$\{(x,y),(x+1,y-1),(x,y-1),(x,y-2)\} \tag{22}$$

$$\{(x,y),(x-1,y),(x-1,y+1),(x-1,y-1)\}. \tag{23}$$

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

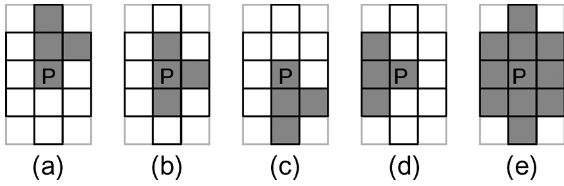


Fig. 12. Method of making a parallel access pattern.

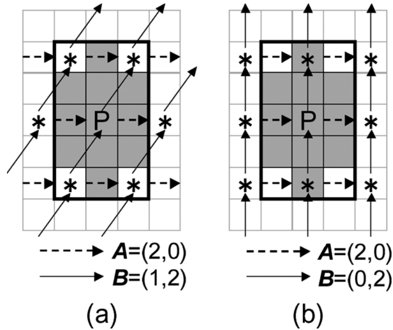


Fig. 13. Parallel access check for a single window. (a) Allocation1. (b) Allocation2.

By finding the union of these sets, the parallel access pattern of P is expressed as

$$\{(x,y),(x,y+2),(x-1,y+1),(x,y+1),\\ (x+1,y+1),(x-1,y),(x+1,y),(x-1,y-1),\\ (x,y-1),(x+1,y-1),(x,y-2)\}. \quad (24)$$

This parallel access pattern is shown in Fig. 12(e).

By using the parallel access pattern, we improve $CHECK_PARALLEL_ACCESS$. Let us consider the function $CHECK_PARALLEL_ACCESS$ when the window shown in Fig. 6(a) is given. First, let us make a parallel access pattern for an arbitrary pixel P as shown in Fig. 12. Next, let us make a memory allocation using a vector pair $(\boldsymbol{A}, \boldsymbol{B})$ in the minimum rectangle which includes the parallel access pattern. Fig. 13(a) shows a result of a memory allocation using $\boldsymbol{A} = (2,0)$ and $\boldsymbol{B} = (1,2)$. Fig. 13(b) shows a result of a memory allocation using $\boldsymbol{A} = (2,0)$ and $\boldsymbol{B} = (0,2)$. The pixels labeled as "$*$" are allocated to the same memory module as P. A memory allocation enables parallel access if these pixels are not included in the parallel access pattern. The memory allocation shown in Fig. 13(a) is capable of parallel access since all pixels labeled as "$*$" are not included in the parallel access pattern. On the other hand, the memory allocation shown in Fig. 13(b) is not capable of parallel access since two pixels labeled as "$*$" are included in the parallel access pattern, that is, gray pixels.

For multiple windows, we make a parallel access pattern for each window. Like the case of a single window, the parallel access check for each window is done using its parallel access pattern. The memory allocation enables parallel access if the parallel access is possible for each window. We consider the example of two windows shown in Fig. 14(a) and (b). Fig. 14(c) and (d) show the parallel access patterns for the windows: W_1 and W_2, respectively. Fig. 15(a) shows a result

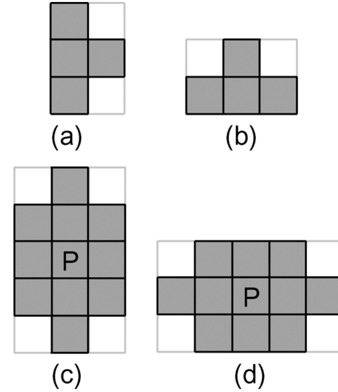


Fig. 14. Parallel access pattern for multiple windows. (a) W_1. (b) W_2. (c) Parallel access pattern for W_1. (d) Parallel access pattern for W_2.

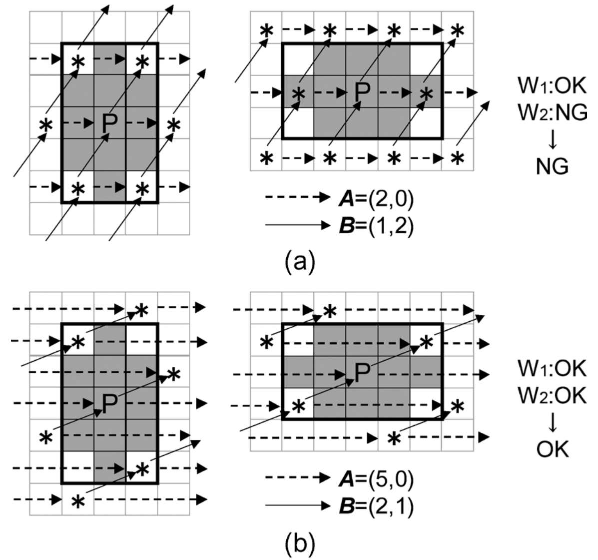


Fig. 15. Parallel access check for multiple windows. (a) Parallel access check for $\boldsymbol{A} = (2,1)$, $\boldsymbol{B} = (1,2)$. (b) Parallel access check for $\boldsymbol{A} = (5,0)$, $\boldsymbol{B} = (2,1)$.

of a memory allocation of $\boldsymbol{A} = (2,0)$ and $\boldsymbol{B} = (1,2)$ for these parallel access patterns. Fig. 15(b) shows a result of a memory allocation of $\boldsymbol{A} = (5,0)$ and $\boldsymbol{B} = (2,1)$. The memory allocation shown in Fig. 15(a) is not capable of parallel access since two pixels labeled as "$*$" are included in the parallel access pattern for W_2. The memory allocation shown in Fig. 15(b) is capable of parallel access since no pixel labeled as "$*$" is included in both parallel access patterns.

By using the previous method, the range to be checked is limited to around the parallel access pattern for a single arbitrary pixel P. Therefore, we can reduce the time complexity of the $CHECK_PARALLEL_ACCESS$.

The proposed algorithm can always find a vector for multiple windows as follows. Given multiple windows, the most simple way of finding the memory allocation for parallel access is to apply the rectangular memory allocation [10] (or [9]) to the minimum rectangle window that includes all the given windows. The concept behind this method is to approximate the multiple windows by a single circumscribing rectangle window. Although this method can require the large number of memory

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

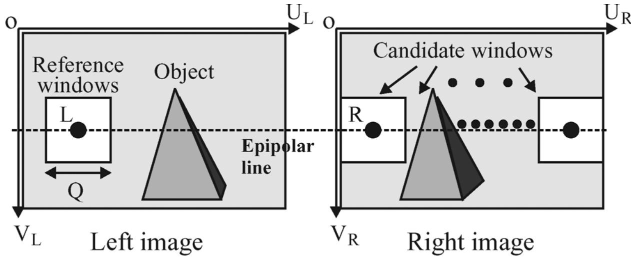


Fig. 16. Search for a corresponding point.

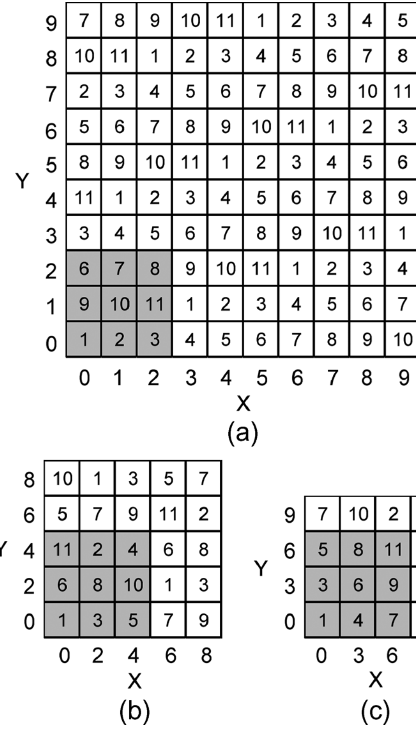


Fig. 17. Optimal memory allocation for multi-resolution images ($Q = 3$ and $\mathrm{SP}_{\max} = 3$). (a) Original image ($\mathrm{SP} = 1$). (b) Reduced image ($\mathrm{SP} = 2$). (c) Reduced image ($\mathrm{SP} = 3$).

modules when the approximation is not good, this guarantees the parallel access. The larger rectangle window intuitively corresponds to the larger equivalent vector pair in our algorithm. Our algorithm begins with the possible largest equivalent vector pair, that is, the possible largest rectangular window. Then, our algorithm iteratively improves the current optimal solution by shortening the length of the equivalent vector pair. Therefore, our algorithm can always find the memory allocation for parallel access with multiple windows.

IV. Design Examples

We show two examples where multiple windows are efficiently used and frequently appear in real cases. In these examples, pixels used at the same time are sparsely arranged, and the conventional memory allocation methods require a large number of memory modules.

Section IV-A describes stereo matching using multi resolution images. The image processing is not limited to stereo matching. It frequently appears in image processing such as recognition and so on.

Section IV-B describes optical flow extraction using a single square window. Use of a single square window is very popular in image processing such as filtering and so on. The image processing using a square window is attributed to image processing using multiple windows when pixel data in an overlapping area are reused.

A. Stereo Matching VLSI Processor Using Multi-Resolution Images

Stereo matching is one efficient method to obtain 3-D information of a real scene. It uses two images taken from two different cameras at the same time. After correspondence between the two images is established, the 3-D information of the scene is computed based on the triangular method. Given a reference window, SADs are computed for all the possible candidate windows as shown in Fig. 16. Note that reference and candidate windows have a square shape. The candidate window with the minimum SAD is determined to be the corresponding pixel.

Multi-resolution images are used to reduce the computational amount. Given sampling periods $\mathrm{SP} = \mathrm{SP}_{\max}, \mathrm{SP}_{\max-1}, \ldots, 2, 1$, reduced images are made by sampling the original images every SP pixels. Beginning with the lowest-resolution image ($\mathrm{SP} = \mathrm{SP}_{\max}$), the resolution is iteratively increased until $\mathrm{SP} = 1$. The possible locations of candidate windows at a higher resolution are limited by using the matching result at a lower resolution. Hence, the computational amount is reduced [11].

Fig. 17 shows the resulting optimal memory allocation for $Q = 3$ and $\mathrm{SP}_{\max} = 3$. The period vectors are $\boldsymbol{A} = [11\ 0]^T$ and $\boldsymbol{B} = [3\ 1]^T$. The number of memory modules is 11. The label $k (= 1, 2, \ldots, 11)$ on a pixel denotes which memory module the pixel is allocated to. Figs. 17(a)–(c) correspond to $\mathrm{SP} = 1$, 2, and 3, respectively. Gray pixels are accessed in parallel at each sampling period. You can see that all the pixels in the window are distributed among different memory modules for arbitrary locations of the window. In other words, parallel memory access for a window is enabled at each sampling period.

Let us compare this result with that of three conventional methods.

The first method is the rectangular memory allocation [10]. This is originally used for image processing with a single resolution. The rectangular memory allocation maps pixels in a rectangular window onto different memory modules, where the rectangular window is defined as the minimum rectangular one that can include all the parallel-accessed pixels. In the example shown in Fig. 17, the rectangular window is determined such that it includes all the gray pixels shown in Fig. 17(c). Hence, the rectangular memory allocation requires a 7×7 rectangular window with $49 (= 7 \times 7)$ memory modules as shown in Fig. 18(a). The gray pixels in Fig. 18(a) correspond to the gray ones in Fig. 17(c).

The second one is the tile-based memory allocation [9]. This is originally used for image processing with a single resolution. Note that the tile-based allocation [9] provides the same result as the rectangular one [10].

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

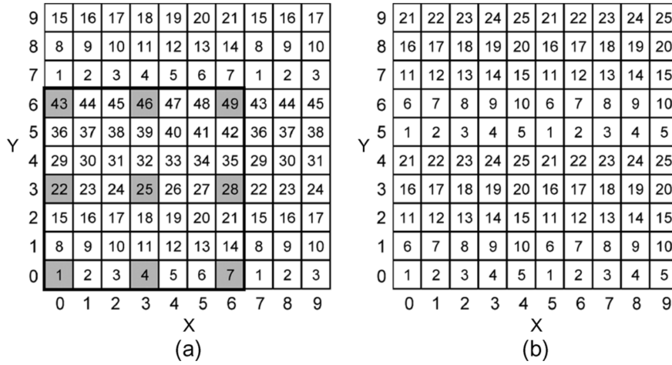


Fig. 18. Conventional memory allocation ($Q = 3$ and $\mathrm{SP}_{\max} = 3$). (a) Rectangular memory allocation (original image). (b) Memory allocation with limited period vectors (original image).

The third one is used for image processing with multi-resolution as well as the proposed method [11]. This method differs from the proposed method in that periods vectors are limited in terms of length and direction as follows.

- Periods vectors $\boldsymbol{U}$ and $\boldsymbol{V}$ have the same length.
- $\boldsymbol{U}$ and $\boldsymbol{V}$ are a horizontal vector and a vertical one, respectively.

We call this method memory allocation with limited period (MALP) vectors. Fig. 18(b) shows the result of MALP for $Q = 3$ and $\mathrm{SP}_{\max} = 3$. This memory allocation requires $25(= 5 \times 5)$ memory modules. The period vectors $\boldsymbol{U}$ and $\boldsymbol{V}$ for this allocation are given by

$$\boldsymbol{U} = [5\ 0]^T, \boldsymbol{V} = [0\ 5]^T. \tag{25}$$

As a result, the number of memory modules of the proposed method is reduced to $23\%(= 11/49)$ of the rectangular memory allocation, $23\%(= 11/49)$ of the tile-based one, and $44\%(= 11/25)$ of MALP.

For the practical case, the window size $Q = 4$, and the maximum sampling period $\mathrm{SP}_{\max} = 8$ [11]. The proposed memory allocation, the rectangular one [10], the tile-based one [9] and MALP [11] require 17, $625(= 25^2)$, $625(= 25^2)$, and $121(= 11^2)$ memory modules, respectively. The search time of the proposed method is less than 1 ms on a PC (Pentium4 at 2 GHz, 1.2 GB main memory, OS: Windows XP). The total memory capacity of memory modules is constant independently of the number of memory modules from the condition C3. The number of AD units and adders depends on the number of memory modules. The total number of AD units and adders of the proposed memory allocation is reduced to 2.7%, 2.7%, and 14% in comparison with that of three conventional allocation methods, respectively. Table II summarizes the comparison results. Image size is 500 × 500 pixels with 256-level grayscale. The upper and lower parts denote the architectural results and the results of FPGA-based implementations, respectively. The StratixIII (EP3SE260F1152C3) is used for the implementations. Altera's Quartus II is used for simulation, mapping, and power analysis. The processing time is evaluated at the maximum frequency. The comparison result does not include the operation steps to load pixels into the memory modules. The operation steps to load pixels are same in the proposed and the conventional methods since the number of pixels is same. The power dissipation is evaluated at 60 MHz since the processor based on the rectangular allocation runs at less than 64 MHz. The CAD "Quartus II" simulates considering the real configuration data using real image inputs. Therefore, the resulting processing time and power dissipation will be to very close to the real values.

TABLE II
COMPARISON RESULTS FOR THE STEREO-MATCHING PROCESSOR

	Proposed allocation	MALP[11]	Rectangular allocation[10]	Tile-based allocation[9]
Number of memory modules	17	121	625	625
Number of AD untis	17	121	625	625
Number of adders	16	120	624	624
LUTs	7,951	26,241	106,327	106,327
Registers	3,944	4,134	15,180	15,180
Interconnection usage[%]	1	7	40	40
Maximum Frequency[MHz]	129	91	64	64
Processing time[ms](@Maximum Frequency)	9.8	13.8	19.7	19.7
Total power dissipation[mW](@60MHz)	1,248	1,324	1,679	1,679

StratixIII (EP3SE260F1152C3) FPGA is used.

B. Optical Flow

Given an image sequence, matching between continuous two images is frequently used in many applications such as motion estimation for data compression, optical flow, and so on. Fig. 19 illustrates the typical situation of the image matching between continuous two images. Figs. 19(a) and (b) are images at times T and $T + dT$, respectively. The image at time T is called a reference image, and the image at time $T + dT$ a candidate image. We consider a square window called a reference window of a predetermined size $E \times E$ in the reference image. In order to find the corresponding window in the candidate image, the similarity measure such as an SAD is computed between the reference window and a candidate one for all possible locations of the candidate window within a search area. The sequence of the locations provides a great impact on how many pixels should be accessed in parallel. If the consecutive candidate windows do not overlap to each other, $E \times E$ pixels must be newly retrieved. To minimize the number of newly-retrieved pixels, the consecutive candidate windows must have a maximum overlap. Fig. 19(b) shows the sequence of the locations of the candidate windows to meet the requirement. In this square-wave-like sequence, the candidate window moves horizontally or vertically by one pixel. For example, Fig. 20 shows the newly-retrieved pixels for a candidate window of size 3 × 3. When the candidate window moves horizontally from L1 to L2 as shown in Figs. 20(b) and (c), pixels $C_{2,1}$, $C_{2,2}$, $C_{2,3}$, $C_{3,1}$, $C_{3,2}$, and $C_{3,3}$ can be reused by storing them into registers. Pixels $C_{4,1}$, $C_{4,2}$, and $C_{4,3}$ must be newly retrieved from memory modules. Similarly, when the candidate window moves vertically from L1 to L3 as shown in Figs. 20(b) and (d), pixels $C_{1,2}$, $C_{2,2}$, $C_{3,2}$, $C_{1,3}$, $C_{2,3}$, and $C_{3,3}$ can be reused. Pixels $C_{1,4}$, $C_{2,4}$, and $C_{3,4}$ must be newly retrieved from memory modules. In general, for the candidate window of size $E \times E$, the square-wave-like sequence requires parallel access for only E pixels either in a column or a row of the candidate window although $E \times E$ pixels are used for matching.

The windows for memory allocation, W1 and W2 are of size $E \times 1$ or $1 \times E$ as shown in Fig. 21(a) and (b), respectively. Note that the window size for memory allocation is smaller

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

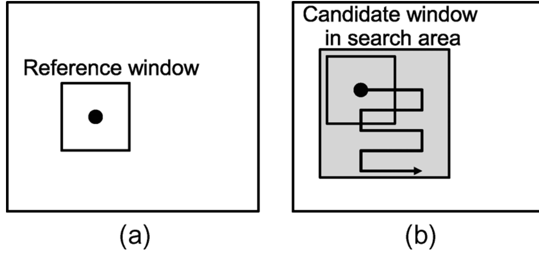


Fig. 19. Optical flow. (a) Image at time T (reference image). (b) Image at time $T + dT$ (candidate image).

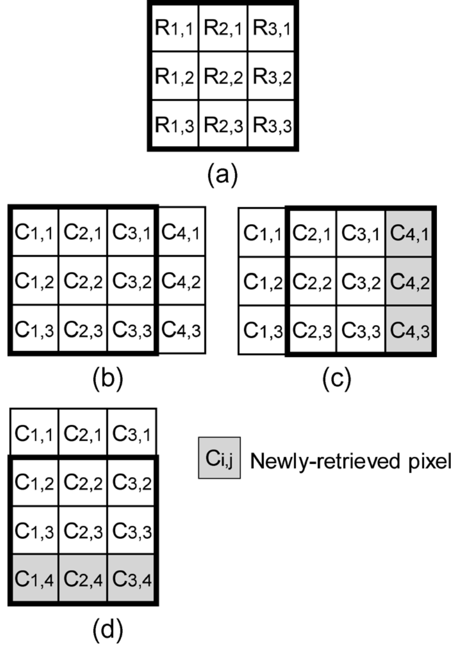


Fig. 20. Newly-retrieved pixel for optical flow computation. (a) Reference window. (b) Candidate window at location L1. (c) Candidate window at location L2. (d) Candidate window at location L3.

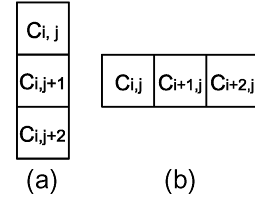


Fig. 21. Windows for memory allcation. (a) Window W1. (b) Window W2.

5	4	3	2	1	10	9	8	7	6	5	4	3	2	1
4	3	2	1	10	9	8	7	6	5	4	3	2	1	10
3	2	1	10	9	8	7	6	5	4	3	2	1	10	9
2	1	10	9	8	7	6	5	4	3	2	1	10	9	8
1	10	9	8	7	6	5	4	3	2	1	10	9	8	7
10	9	8	7	6	5	4	3	2	1	10	9	8	7	6
9	8	7	6	5	4	3	2	1	10	9	8	7	6	5
8	7	6	5	4	3	2	1	10	9	8	7	6	5	4
7	6	5	4	3	2	1	10	9	8	7	6	5	4	3
6	5	4	3	2	1	10	9	8	7	6	5	4	3	2
5	4	3	2	1	10	9	8	7	6	5	4	3	2	1
4	3	2	1	10	9	8	7	6	5	4	3	2	1	10
3	2	1	10	9	8	7	6	5	4	3	2	1	10	9
2	1	10	9	8	7	6	5	4	3	2	1	10	9	8
1	10	9	8	7	6	5	4	3	2	1	10	9	8	7

Fig. 22. Optimal memory allocation for optical flow.

TABLE III
COMPARISON RESULT FOR THE OPTICAL-FLOW-EXTRACTION PROCESSOR

	Proposed allocation	Rectangular allocation[10]	Tile-based allocation[9]
Number of memory modules	10	100	100
Number of AD untis	100	100	100
Number of adders	99	99	99
LUTs	8,840	94,339	94,339
Registers	487	239	239
Interconnection usage[%]	2	37	37
Maximum Frequency[MHz]	178	76	76
Processing time[ms](@Maximum Frequency)	153	359	359
Total power dissipation[mW](@75MHz)	1,039	3,119	3,119

StratixIII (EP3SL200F1152C2) is used.

than that for optical flow since the pixels are reused as mentioned before. Fig. 22 shows the resulting optimal memory allocation for W1 and W2 for $E = 10$. The optimal memory allocation requires 10 memory modules. Pixels in a diagonal line are allocated to the same memory module, and this memory allocation is called "diagonal memory allocation". The resulting number of memory modules is minimum since each of W1 and W2 have 10 pixels for $E = 10$. The search time of the proposed method is less than 1 ms on a PC (Pentium4 at 2 GHz, 1.2 GB main memory, OS: Windows XP). Let us compare the diagonal memory allocation with the conventional rectangular memory allocation [10] and the tile-based one [9]. The rectangular memory allocation requires $E \times E$ memory modules since the minimum rectangle including both of W1 and W2 is of size $E \times E$. For $E = 10$, the number of memory modules for the diagonal memory allocation is reduced to $10\%(= 10/100)$ of that for the rectangular one, and $10\%(= 10/100)$ of that for the tile-based one. Table III summarizes the comparison results. Image size is 500×500 pixels with 256-level grayscale. Search area is 10×10 pixels. Note that MALP [11] is not compared with the proposed method since MALP can be used only for multi-resolution images. The upper and lower parts denote the architectural results and the results of FPGA-based implementations, respectively. The StratixIII (EP3SL200F1152C2) is used for the implementations. This FPGA is the smaller than that used in the stereo-matching processor. The processing time is evaluated at the maximum frequency. The comparison result does not include the operation steps to load pixels into the memory modules. The operation steps to load pixels are same in the proposed and the conventional methods since the number of pixels is same. The power dissipation is evaluated at 75 MHz since the processor based on the rectangular allocation runs at less than 76 MHz.

V. DISCUSSION

From the result of Section X, there is the large variance among the reduction ratios of the numbers of memory modules for the two examples: 2.7% for the stereo-matching processor and 10% for optical-flow extraction processor, respectively. These reduction ratios are defined as (the number of memory

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

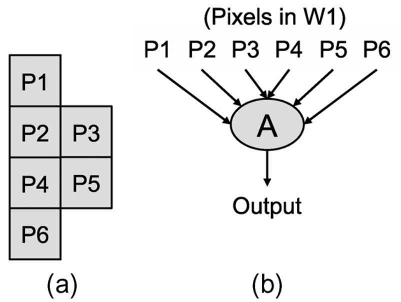


Fig. 23. Window and SDFG for single-step operation. (a) Window W1. (b) SDFG for W1.

(a)

2	1	2	1	2	1	2	1	2	1
5	6	5	6	5	6	5	6	5	6
3	4	3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2	1	2
6	5	6	5	6	5	6	5	6	5
4	3	4	3	4	3	4	3	4	3
2	1	2	1	2	1	2	1	2	1
5	6	5	6	5	6	5	6	5	6
3	4	3	4	3	4	3	4	3	4
1	2	1	2	1	2	1	2	1	2

(b)

2	6	2	6	2	6	2	6	2	6
1	5	1	5	1	5	1	5	1	5
4	8	4	8	2	8	2	8	2	8
3	7	3	7	3	7	3	7	3	7
2	6	2	6	2	6	2	6	2	6
1	5	1	5	1	5	1	5	1	5
4	8	4	8	4	8	4	8	4	8
3	7	3	7	3	7	3	7	3	7
2	6	2	6	2	6	2	6	2	6
1	5	1	5	1	5	1	5	1	5

Fig. 24. Memory allocation result for W1. (a) Proposed memory allocation. (b) Rectangular memory allocation [10]/tile-based memory allocation [9].

modules of the our method)/(the number of memory modules of the rectangular memory allocation). The reduction ratio intuitively denotes how well the rectangular window approximates the given window/windows. The better the rectangular window approximates the given window/windows, the larger the reduction ratio is. In the examples of the stereo-matching processor and the optical-flow extraction processor, the circumscribing rectangular window does not approximate the given windows well, and is sparsely occupied by the pixels of the given window. By exploiting the sparseness, our algorithm shrinks the equivalent vector pair to reduce the number of memory modules.

We can relax the constraint of the window-serial-and-pixel-parallel scheduling where a window-operation is executed in a single step, and all the pixels in a window must be retrieved in parallel. For more practical cases, we should consider multi-step operation where a window-operation is executed in multiple steps, and pixels in a window are also retrieved in multiple steps. Fig. 23 shows a window and the SDFG for single-step operation based on the window-serial-and-pixel-parallel scheduling. All the pixels in W1 must be accessed in parallel to meet the time constraint imposed by the SDFG. Fig. 24(a) and (b), respectively, show the proposed memory allocation and the rectangular memory allocation for the window W1, which respectively require 6 and 8 memory modules. Note that the tile-based allocation [9] provides the same result as the rectangular memory allocation [10] as mentioned before.

Fig. 25 shows windows and the SDFG for multi-step operation (Type 1). To relax the window-serial-and-pixel-parallel scheduling, W1 is divided into W2 and W3, and pixels in either

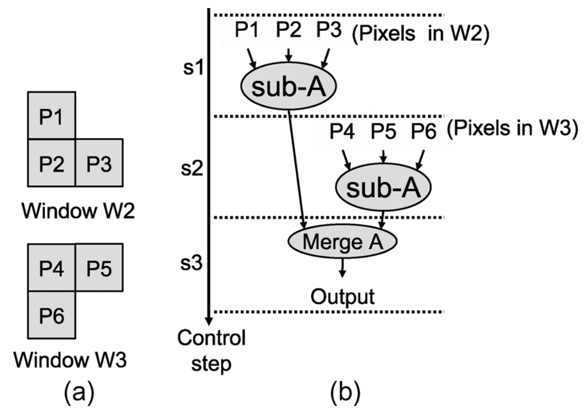


Fig. 25. Window and SDFG for multi-step operation (Type 1). (a) Windows. (b) SDFG for W2 and W3.

3	4	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4	1	2
3	4	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4	1	2
3	4	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4	1	2
3	4	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4	1	2
3	4	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4	1	2

Proposed memory allocation and Rectangular memory allocation[10] / tile-based memory allocation[9]

Fig. 26. Memory allocation result for multi-step operation (Type 1).

W2 or W3 are accessed in parallel at a single step. The label "Sub-A" denotes a partial operation of operation of the type "A". At steps S1 and S2, Sub-A for W2 and W3 are performed, respectively. At step S3, an additional operation is required to merge the results of the Sub-A operations. For example, if operation of the type A means 6-input addition, Sub-A and Merge-A mean 3-input addition and 2-input addition, respectively. The memory allocation problem for the multi-step operation (Type 1) shown in Fig. 25 can be considered to be the memory allocation problem for multiple windows W2 and W3. This is because pixels in W2 or W3 must be accessed in parallel for any locations of them. Hence, the memory allocation for multi-step operation (Type 1) can be solved by the proposed method for multiple windows. Fig. 26 shows the optimal memory allocation for the windows W2 and W3, which requires four memory modules.

The reduction ratio for a multi-step operation depends on not only a memory allocation method but also a manner of division of a window. For the example of multi-step operation (Type 1) shown in Fig. 25, the same memory allocation shown in Fig. 26 is obtained by either the proposed memory allocation or rectangular one/tile-based one. On the other hand, for the example of multi-step operation (Type 2) shown in Fig. 27, the proposed memory allocation is more efficient, that is, provides a less reduction ratio. In this case, W1 is divided into W4 and W5, and pixels in either W4 or W5 are accessed in parallel at a single

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

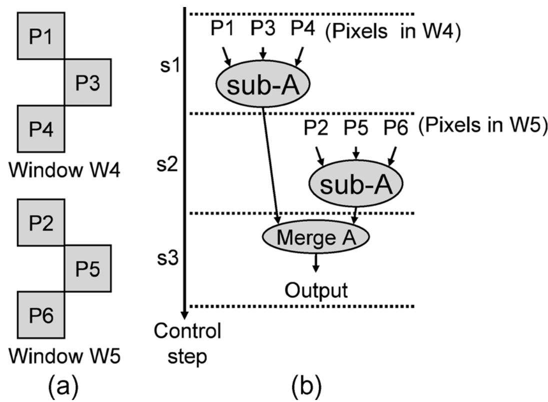


Fig. 27. Window and SDFG for multi-step operation(Type 2). (a) Windows. (b) SDFG for W4 and W5.

(a)

1	1	1	1	1	1	1	1	1	1
3	3	3	3	3	3	3	3	3	3
2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1
3	3	3	3	3	3	3	3	3	3
2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1
3	3	3	3	3	3	3	3	3	3
2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1

(b)

1	4	1	4	1	4	1	4	1	4
3	6	3	6	3	6	3	6	3	6
2	5	2	5	2	5	2	5	2	5
1	4	1	4	1	4	1	4	1	4
3	6	3	6	3	6	3	6	3	6
2	5	2	5	2	5	2	5	2	5
1	4	1	4	1	4	1	4	1	4
3	6	3	6	3	6	3	6	3	6
2	5	2	5	2	5	2	5	2	5
1	4	1	4	1	4	1	4	1	4

Fig. 28. Memory allocation results for multi-step operation (Type 2). (a) Proposed memory allocation. (b) Rectangular memory allocation [10]/tile-based memory allocation [9].

step. Fig. 28(a) and (b) respectively show the proposed memory allocation and the rectangular one/tile-based one for the windows W4 and W5, which respectively require 3 and 6 memory modules. Note that the number of memory modules required by the proposed memory allocation does not exceed that required by the rectangular one/tile-based one even in the worst case.

VI. CONCLUSION

This paper presents an optimal memory allocation method to minimize the number of memory modules under a time constraint. The method is also useful for FPGA implementations where interconnect overhead in delay is significant large.

The architecture model is one simplified form of recent image processors [1], [2]. These processors are based on highly-parallel SIMD architecture where each PE has its own local memory module like our architecture model. [1] and [2] have 128 and 256 pairs of a PE and a memory module, respectively. The major difference between our architecture and them is that they have more complex inter-PE network such as linear array. They are designed for power-aware embedded applications such as digital cameras, mobile phone and advanced safe vehicles. Their power consumption is less than 3 W. Nowadays, the use of graphics processing units (GPUs) is also one efficient solution to accelerate image processing because of their high degree of parallelism [19], [20].[1] For example, the NVIDIA Geforce6800 [21] has 6 vertex processors with the MIMD manner, 12 fragment processors with the SIMD manner, 4 memory modules, and a crossbar network between the vertex and fragment processors. More advanced GPUs such as RADEON X1900XT, GeForce7800, and RADEON X800 have more vertex and fragment processors. The state-of-art GPUs can be considered as a multi-processor where each processor has a local memory. Such architecture is similar to our target architecture. Therefore, the proposed memory allocation is applied to image processing using the state-of-art GPUs. GPUs are mainly used not for implementing final products but for evaluating algorithms, because of their large power consumptions. Their power consumptions range from 30 to 100 W. To exploit the parallelism of the recent image processors and GPUs, memory allocation for parallel access is essential. Our memory allocation can be applied to them. In such cases, the advantage of minimizing the number of memory modules is to save the dynamic power consumed by unused PEs and memory modules, not to minimize the chip area. If power gating is available, it is also possible to save the static power consumption of the unused PEs and memory modules.

REFERENCES

[1] S. Kyo, T. Koga, S. Okazaki, and I. Kuroda, "A 51.2-GOPS scalable video recognition processor for intelligent cruise control based on a linear array of 128 four-way VLIW processing elements," *IEEE J. Solid-State Circuits*, vol. 38, no. 11, pp. 1992–2000, Nov. 2003.

[2] M. Nakajima, H. Noda, K. Dosaka, K. Nakata, M. Higashida, O. Yamamoto, K. Mizumoto, H. Kondo, Y. Shimazu, K. Arimoto, K. Saitoh, and T. Shimizu, "A 40GOPS 250 mW massively parallel processor based on matrix architecture," in *Proc. ISSCC Dig. Tech. Papers*, Feb. 2006, pp. 410–411.

[3] L. Ramachandran, D. Gajski, and V. Chaiyakul, "An algorithm for array variable clustering," in *Proc. Euro. Des. Autom. Conf.*, 1994, pp. 262–266.

[4] N. Holmes and D. Gajski, "Architectural exploration for datapaths with memory hierarchy," in *Proc. Euro. Des. Autom. Conf.*, 1994, pp. 340–344.

[5] P. R. Panda, "Memory bank customization and assignment in behavioral synthesis," in *Proc. Int. Conf. Comput.-Aided Des.*, 1999, pp. 477–481.

[6] W. T. Shiue, S. Tadas, and C. Chakrabarti, "Low power multi-modules, multi-port memory design for embedded systems," in *Proc. Signal Process. Syst.*, 2000, pp. 529–538.

[7] S. Wuytack, F. Catthoor, D. De Jong, and H. De Man, "Minimizing the required memory bandwidth in VLSI system relizations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 4, pp. 433–441, Dec. 1999.

[8] J. Seo, T. Kim, and P. R. Panda, "Memory allocation and mapping in high-level synthesis—An integrated approach," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, no. 5, pp. 928–938, May 2003.

[9] P. Ranjan, Panda, and N. D. Dutt, "Low-power memory mapping through reducing address bus activity," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 3, pp. 309–320, Sep. 1999.

[10] M. Hariyama, S. Lee, and M. Kameyama, "Highly-parallel stereo vision VLSI processor based on an optimal parallel memory access scheme," *IEICE Trans. Electron.*, vol. E84-C, no. 3, pp. 382–389, 2001.

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

[12] D. Marr and E. Hildreth, "Theory of edge detection," *Proc. Royal Society London B*, vol. 207, pp. 187–217, 1980.

[13] A. Kaced, "The K-forms: A new technique and its applications in digital image processing," in *Proc. IEEE 5th Int. Conf. Pattern Recognition*, 1980, pp. 933–936.

[1][Online]. Available: http://www.gpgpu.org

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.

[14] H. Minkowski, “Volumen und oberflache,” *Math. Annalen*, vol. 57, pp. 447–495, 1903.
[15] J. Klein and J. Serra, “The texture analyzer,” *J. Microscopy*, vol. 95, pp. 349–356, 1972.
[16] R. Haralick, S. Sternberg, and X. Zhuang, “Image analysis using mathematical morphology: Part I,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 9, no. 4, pp. 532–550, Jul. 1987.
[17] K. Sivakumar and J. Goutsias, “Morphologically constrained grfs: Applications to texture synthesis and analysis,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 2, pp. 99–131, Feb. 1999.
[18] H. Heijmans, “Theoretical aspects of gray-level morphology,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 6, pp. 568–582, Jun. 1991.
[19] N. Cornelis, “Real-time connectivity constrained depth map computation using programmable graphics hardware,” in *Proc. CVPR*, 2005, vol. 1, pp. 1099–1104.
[20] S. Shinha, J.-M. Frahm, and M. Pellefeys, “GPU-based video feature tracking and matching,” Univ. North Calolina, Chapel Hill, Tech. Rep. TR06-012, May 2006.
[21] J. Montrym and H. Moreton, “The GeForce 6800,” *IEEE Micro*, vol. 25, no. 2, pp. 41–51, Feb. 2005.

Yasuhiro Kobayashi (M’06) received the B.E. degree in electronic engineering from Tohoku University, Sendai, Japan, in 1997.

He is currently a technical staff with Oyama National College of Technology, Oyama, Japan. His research interests include reconfigurable VLSIs for computer vision.

Masanori Hariyama (M’02)received the B.E degree in electronic engineering, the M.S. degree in information sciences, and the Ph.D. degree in information sciences from Tohoku University, Sendai, Japan, in 1992, 1994, and 1997, respectively.

He is currently an Associate Professor with the Graduate School of Information Sciences, Tohoku University, Sendai, Japan. His research interests include VLSI computing for real-world application such as robots, high-level design methodology for VLSIs and reconfigurable computing.

Michitaka Kameyama (M’79–F’97) received the B.E., M.E., and D.E. degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1973, 1975, and 1978, respectively.

He is currently a Professor with the Graduate School of Information Sciences, Tohoku University. His general research interests include intelligent integrated systems for real-world applications and robotics, advanced VLSI architecture, and new-concept VLSI including multiple-valued VLSI computing.

Prof. Kameyama was a recipient of 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.

Authorized licensed use limited to: TOHOKU UNIVERSITY. Downloaded on March 10,2010 at 02:29:01 EST from IEEE Xplore. Restrictions apply.