Available online at www.sciencedirect.com
SCIENCE DIRECT®
Signal Processing: *Image Communication* 19 (2004) 975–992
SIGNAL PROCESSING: IMAGE COMMUNICATION
www.elsevier.com/locate/image

A simple and efficient block motion estimation algorithm based on full-search array architecture

Shih-Yu Huang[a,*], Wei-Chang Tsai[b]

[a]*Department of Computer Science and Information Engineering, Ming Chuan University, 5 Teh-Ming Rd., Gwei Shan District, Taoyuan Country 333, Taiwan*

[b]*National Chip Implementation Center, National Applied Research Laboratories, 1F, No. 1, Prosperity RD. I, Science-Based Industrial Park, Hisnchu 300, Taiwan*

Received 25 February 2004

Abstract

This paper first presents an array structure using ±1 full-search (FS) architecture as the search engine of block motion estimation which takes advantage of the design regularity of FS. An efficient algorithm named modified gradient-descent search (MGDS) is then introduced based on the proposed architecture. MGDS utilizes an adaptive computation distribution mechanism to efficiently allocate computation of the employed ±1 FS array to blocks or frames of video sequences. Experimental results indicate that MGDS uniformly achieves a higher quality than FS by an amount that is dependent on motion activities of sequences.

Keywords: Motion estimation; Block matching; Video coding

1. Introduction

Video compression standards such as ISO MPEG-1, -2, and -4, and ITU-T H.261, 262, 263, and 264 [4–6,13] use block-based motion compensation techniques to eliminate temporal redundancy in video sequences. The block matching algorithm (BMA) is extensively employed to extract motion vectors (MVs). Typically, the algorithm consumes 60–80% of the total computation in a video encoder, and it strongly affects the visual quality at a given bit-rate. The regular data flow of full-search (FS) BMA makes it especially amenable to hardware implementation. Many efficient hardware designs have been proposed for FS in recent decades [1,3,8,10,12], with many focusing on data reuse. However, FS—a brute-force algorithm—does not utilize information on motion activity in video sequences, and hence it is possible to improve the performance of

*Corresponding author. Tel.: +886-3-3507001.
E-mail addresses: syhuang@mcu.edu.tw (S.-Y. Huang), wctsai@cic.org.tw (W.-C. Tsai).

0923-5965/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.image.2004.08.001

FS hardware designs by considering this issue. Based on the center-biased characteristics of MVs [2], several fast BMAs have been developed, including the three-step search (TSS) [7], the new TSS (NTSS) [9], the block-based gradient-descent search (GDS) [11], the diamond search (DS) [16], and PMVFAST [14]. Although these fast BMAs greatly reduce the computation required for motion estimation, their irregular search patterns result in complicated hardware design.

An algorithmic and architectural codesign of a motion estimation engine was proposed in 2002 [15]. Four rules were considered in the design of BMAs. First, searching points should be chosen in the direction of the current best improvement for faster convergence to an optimum solution, i.e., GDS. Second, the spatial and temporal correlation of MVs should be exploited to determine the initial searching point. Third, searching points should be examined in a pattern around the initial position so as to exploit the center-biased distribution of MVs. Finally, the search should stop as soon as possible once the matching is good enough. Based on the above principles, a directional squared-search (DSS) algorithm and a pipelined parallel architecture are presented.

The core of DSS follows the first rule, as shown in Fig.1. A square 3×3 search window of nine points is initially applied to the search area with the center recommended by the second rule. The algorithm stops if the center of the 3×3 window is the position of the best matching point; otherwise,

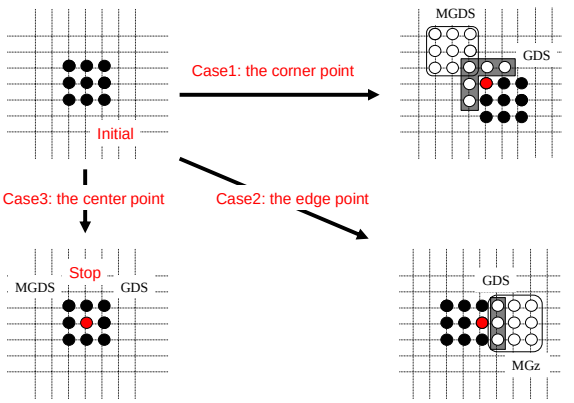

Fig. 1. Structure of GDS and MGDS.

the search center is moved to the best matching point. Only neighboring points of the center position are investigated in the next search step. If the best matching point is in the corner, then five additional points should be checked, whereas three points must be checked when the best matching point is an edge point. The above process is repeated until the center of the window is the position of the best matching point.

Typically, the initial search step of GDS can be considered to be a special case of FS with a ± 1 search range, and it can be efficiently implemented in many hardware designs. However, the data flow when investigating neighboring points during subsequent searches is not as regular as during the initial step, and hence special hardware is required [15]. Actually, FS is the best choice for motion estimation from the viewpoint of hardware implementation. The basic idea of this paper, as shown in Fig. 1, is to modify the searching points employed by the subsequent searches of GDS such that they can also be performed by ± 1 FS without redundant computation. Irrespective of whether the best matching point is located at a corner or edge, the modified GDS (MGDS) successively applies a square 3×3 search window of new nine points.

The major advantage of the proposed MGDS scheme is that all operations can be performed by ± 1 FS. Thus, we can employ an array of ± 1 FSs as the search engine of motion estimation. This feature is especially useful for personal visual communication because types of handheld devices that are now being used are various, such as PDAs and handsets. The large variation in the computation power amongst these heterogeneous devices makes conventional BMAs impractical, since their parameters cannot be tuned automatically according to the available computational power. In contrast, the computational power of the search engine can be easily updated by changing the number of elements in the FS array. To efficiently utilize the available computation of the search engine, an adaptive computation distribution mechanism is further presented in this paper. Experimental results indicate that MGDS can uniformly achieve a quality improvement over original FS under the same computation. That is,

the computation distortion (CD) performance of FS hardware designs can be improved when they are assembled into the proposed array structure and cooperate in the manner of MGDS.

The remainder of this paper is organized as follows: Section 2 describes the proposed MGDS algorithm, experimental results are given in Section 3, Section 4 embeds MGDS into the FS array architecture, and some conclusions are drawn in Section 5.

2. Modified gradient-descent search algorithm

Current video codec predicts MV of a given macroblock (MB) based on the MVs of neighboring MBs for efficient entropy coding. However, the search center, (x_c, y_c), of the current MB can only be set as (0,0) for parallel extracting MV in MGDS. A square 3×3 search window of nine points is then applied to the initial center, i.e., (0, 0). In MGDS, distortion is measured by the sum of absolute differences (SAD) due to its lower computation cost. Similar to GDS, MGDS immediately stops if the center is the position of the best matching point; otherwise a series of subsequent searches will be performed toward the best matching point. Unlike GDS, a square 3×3 search window of nine points is applied successively irrespective of whether the best matching point is located at a corner or edge. The center of the next search (x_n, y_n) is generated by

$$(x_n, y_n) = (x_c, y_c) + (3 \times i_c, 3 \times j_c), \quad (1)$$

where (i_c, j_c) is the displacement of the best matching point. Fig. 1 shows the demonstrative structure of subsequent searches in MGDS.

In MGDS, three additional stopping conditions are employed to reduce the computation. Let the immediately preceding search center and the SAD of the best matching point be (x_p, y_p) and SAD_p, respectively. It is pointless examining the searched area if the next search center (x_n, y_n) is equal to one of the previous search centers. For simplicity, only the immediately preceding search center (x_p, y_p) is checked in MGDS. Therefore, MGDS stops if $(x_n, y_n) = (x_p, y_p)$. The second stopping condition is $SAD_p \leqslant SAD_c$, where SAD_c is the best SAD of the current MB. This condition implies that the optimum occurs at the best matching point of the previous nine points. The third stopping condition is $SAD_c \leqslant TH$, where TH is the given threshold. This condition indicates that the current SAD is below an acceptable threshold.

Fig. 2(a) shows an example to illustrate the proposed MGDS. Suppose the value of TH is 350. MGDS first examines nine searching points labeled as ① in the figure, resulting in (1, 0) and 500 as the displacement and SAD of the best matching point, respectively. Since the best matching point is not at the center and the best SAD is

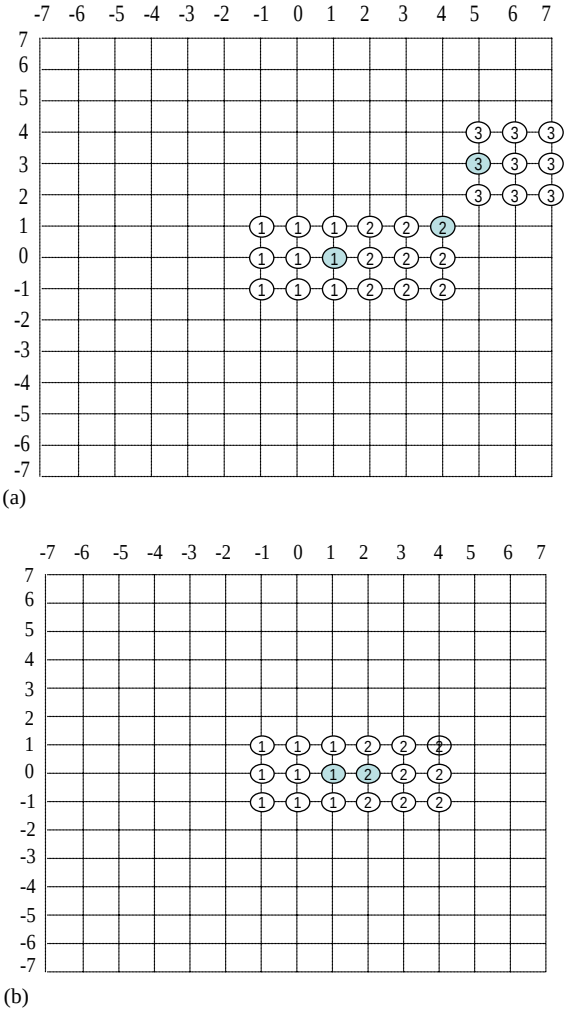

Fig. 2. (a), (b) Examples to illustrate the MGDS algorithm.

larger than TH (i.e., $500 > 350$), the second search center (3,0) is derived using Eq. (1) and nine searching points depicted as ② in the figure are then checked, resulting in (1,1) as the displacement and 400 as the SAD. Similarly, nine searching points labeled as ③ in the figure with the search center (6,3) are examined because the displacement of the best matching point is not (0,0), the current SAD is smaller than that of the previous (i.e., $400 < 500$), the next search center (6,3) is not examined and the best SAD is still larger than TH (i.e., $400 > 350$) in this situation. After the third ± 1 FS, $(i_c, j_c) = (-1, 0)$ and $\mathrm{SAD}_c = 300$. The algorithm then stops because the current best SAD is below TH (i.e., $300 < 350$). Therefore, the MV of this MB is (5,3) since the current SAD is smaller than the previous value. Another example for explaining MGDS is shown in Fig. 2(b). The processing of this example is initially the same as for the above example, except the resulting displacement of the second ± 1 FS is $(-1, 0)$. MGDS stops because the next search center is $(0, 0) = (3, 0) + (3 \times -1, 3 \times 0)$ which has already been examined by the first ± 1 FS. Thus, (2,0) is the final MV.

2.1. Macroblock scheduling for parallel ± 1 full searching

As mentioned, an array of ± 1 FSs is utilized to be the search engine of motion estimation. Thus, these ± 1 FSs can work in parallel to extract MVs. In MGDS, the MBs of a frame are coded in the scan-line order. Once a ± 1 FS is idle, MGDS greedily schedules the next MB to the idled ± 1 FS. The MB will be processed by the scheduled ± 1 FS until one of the stopping conditions is satisfied. Of course, such parallelism prevent using prediction of the initial MV, which is thus forced to be (0,0). The loss of not being able to use initial MV prediction will be discussed in Section 3.

Fig. 3 demonstrates an example of the above scheduling scheme. Assume that the search engine has two ± 1 FSs, denoted as FS(1) and FS(2), and that the MBs in Fig. 2(a) and (b), denoted as MB(1) and MB(2), are the first two MBs to be coded. Each ± 1 FS can process a square 3×3 search window of nine points in one cycle. Since

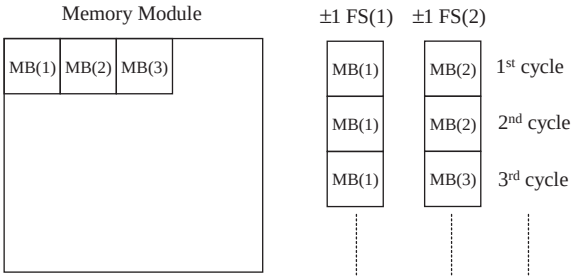

Fig. 3. Example of MB scheduling scheme for parallel full searching.

the two ± 1 FSs are initially all idle, MGDS schedules FS(1) to code MB(1) and FS(2) to code MB(2) in the first cycle. After the first cycle, all stopping conditions are false for MB(1) and MB(2) so that MB(1) and MB(2) are still to be processed by FS(1) and FS(2) in the second cycle. At the end of the second cycle, MB(2) is stopped but MB(1) is not. Therefore, MGDS fetches the next MB to FS(2) and MB(1) is still coded by FS(1) in the third cycle.

2.2. Computation distribution

In general, a codec performs well for videos whose required computation is less than the available computational power. However, the quality of the coded videos will be degraded when the computational power is insufficient to process all the video MBs, such as when input videos contain complex motion. In fact, the quality of coded videos can be improved if the computational power of a codec can be efficiently controlled. This paper proposes two techniques to distribute computational resources.

2.2.1. Explicit computation distribution

The first one is an explicit approach which allocates an upper bound of computation for each MB to control the searches of MGDS. As elsewhere in this paper, the size of computation is in units of searching points. Suppose that SP_c^a and SP_c^p are the numbers of searching points allocated and performed by MGDS of current MB, respectively. This MB should be stopped if the allocated computation is exhausted, i.e.,

$SP_c^p \geqslant SP_c^a$. The method to calculate SP_c^a is presented in the following.

For a frame with N MBs, let T be the total searching points per frame of the employed ± 1 FS array. If the index of the current MB is c, the numbers of uncoded MBs and unused searching points are $N-c+1$ and $T-\sum_{i=1}^{c-1} SP_i^p$, respectively. The simplest method to dynamically regulate the amount of computation is by averaging, for which the computation allocated to the current MB is

$$SP_c^a = \left(T - \sum_{i=1}^{c-1} SP_i^p\right) \Big/ (N-c+1). \tag{2}$$

In general, the video quality is higher when the allocated computation is more precise. A drawback of averaging is that uncoded MBs are all classified into the same type because searching points are equally distributed to them, whereas every uncoded MB requires different computation to extract MV since the corresponding motion activity varies (e.g., little computation is needed for background MBs). A more efficient allocation considers the motion activity of MBs, but it is difficult to determine the motion activity of uncoded MBs because they are not examined in this period. In this paper, uncoded MBs are simply classified into two types, background and non-background, and unused computation is only distributed to uncoded nonbackground MBs. Let N_c^b be the number of uncoded background MBs in the current frame. N_c^b is initially predicted by the number of MBs with MV $= (0,0)$ in the reference frame because successive frames are very similar. Once MGDS generates a (0,0) MV, N_c^b is updated according to $N_c^b = N_c^b - 1$. The computation allocated to the current MB is modified as

$$SP_c^a = \left(T - \sum_{i=1}^{c-1} SP_i^p\right) \Big/ (N-c+1-N_c^b). \tag{3}$$

Computation is not the only constraint for stopping in MGDS, since the current MB will also be stopped even when the allocated computation is not exhausted. The unused computation will be accumulated to code uncoded MBs. Hence, the computation allocated to an MB will tend to increase, which is biased against the earlier MBs. To overcome this problem, a weighting factor $\alpha(\alpha \geqslant 1)$ is used to adjust the value of SP_c^a:

$$SP_c^a = \left[\left(T - \sum_{i=1}^{c-1} SP_i^p\right) \Big/ (N-c+1-N_c^b)\right] \times \alpha. \tag{4}$$

2.2.2. Implicit computation distribution

The third stopping condition of MGDS (i.e., $SAD_c \leqslant TH$) is widely used to reduce the complexity of motion estimation. Actually, the computational power of a codec can also be implicitly controlled by adapting the value of TH, since a small stopping threshold would result in a large number of searching points and vice versa. The advantage of the threshold-adaptation approach is that, given a threshold, the number of searching points computed for a MB directly depends on the accuracy achieved by the motion compensation for this MB. The main challenge of the threshold approach is of course to adapt the threshold so as to keep the total number of searching points below the maximum allowable with the available computational power.

There are usually great similarities among adjacent frames in videos, and we utilize this property to adapt the stopping threshold. Suppose the index of the current frame is k and its stopping threshold is TH_k. MGDS uses TH_k to code all MBs in this frame. The stopping threshold of the next frame, TH_{k+1}, is determined by the results for frame k. The value of the stopping threshold is decreased by a predefined constant Δ if spare computation power is available, i.e., $(T - \sum_{i=1}^{N} SP_i^p) > 0$; otherwise, the value is increased by Δ. This threshold adaptation can be summarized as

$$\begin{cases} TH_{k+1} = \max\{0, TH_k - \Delta\}; \\ \quad \text{if } \left(T - \sum_{i=1}^{N} SP_i^p\right) > 0, \\ TH_{k+1} = TH_k + \Delta; \qquad \text{otherwise.} \end{cases} \tag{5}$$

A suitable value of the stopping threshold is dependent on the motion activity in video sequences and the computational power of the codec. Since we do not have a priori knowledge of motion activity in video sequences, initially it is

assumed that the codec has sufficient computational power for the input videos under consideration. Thus, the initial stopping threshold, i.e., TH_0, is usually set to a small number to achieve higher output quality. The above assumption is of course incorrect, and so the value of the stopping threshold is then automatically adjusted using Eq. (5). The value of Δ determines the sensitivity of the threshold-adaptation mechanism: a large Δ updates the stopping threshold rapidly, and vice versa. Note that $\Delta = 0$ is equivalent to disabling the implicit computation-distribution scheme, i.e., each frame is coded by the original MGDS.

3. Experimental results

In this section, we present experimental results of motion estimation using MGDS, GDS, FS, and TSS. These algorithms are applied to two SIF videos (100 frames each of football and tennis sequences) and two QCIF videos (300 frames each of foreman and salesman sequences). These sequences were selected because their motion characteristics vary considerably, from rapid motion in the football sequence to slow motion in the salesman sequence. The frame rates are set at 15 and 5 Hz to simulate faster and slower networking environments, respectively. The block size is fixed at 16×16 pixels, and the coded quality of motion-compensated videos is measured by the mean square error (MSE) per pixel, which compares the motion-compensated image frames with the original image frames. A lower MSE indicates a smaller prediction error, and a higher quality motion-estimation algorithm.

We first discuss the effects of parallelism in MGDS, which prevents the use of the initial MV prediction. Because all operations are performed by ± 1 FS in MGDS, there are many redundant computations if MGDS likes a standard video codec to examine all MV predictors when determining the best predictor. Accordingly, the initial search center of the compared MGDS is defined as the median of three spatially adjacent MVs (left, top, and top-right). The computation-distribution schemes are first disabled, on the assumption that the available computational power of a codec is sufficient to perform the comparison. Table 1 lists the MSE results of MGDS with and without the initial MV prediction. Although the MSE value of MGDS with an initial MV of (0,0) is worse than that with the above initial MV, the degradation is not so much as a standard video codec selects the best predictor from a set of likely MV predictors. For the worst-case scenario, i.e., the tennis sequence with a 5-Hz frame rate, the MSE values per pixel of MGDS with and without the initial MV prediction are 552.68 and 588.31, respectively (i.e., a 6.45% degradation).

We now describe the experimental results obtained when estimating motion using the proposed computation-distribution schemes on MGDS (the MGDS without computation distribution is also implemented for comparison). In this version, MBs are coded using MGDS until the given computation power is exhausted, after which MVs of uncoded MBs are then set to (0,0). To simulate an environment of heterogeneous devices, the computational power of the employed codec is assumed to range from 3000 to 9000 searching points per frame for the SIF videos, and from 900

Table 1
MSE results of MDGS with/without the initial MV prediction

	Football	Tennis	Foreman	Salesman
(a) 15 Hz				
MDGS without initial MV prediction	451.88	170.24	67.27	8.69
MDGS with initial MV prediction	431.78	167.81	56.95	8.69
(b) 5 Hz				
MDGS without initial MV prediction	1129.70	588.31	271.21	44.66
MDGS with initial MV prediction	1126.24	552.68	242.67	44.53

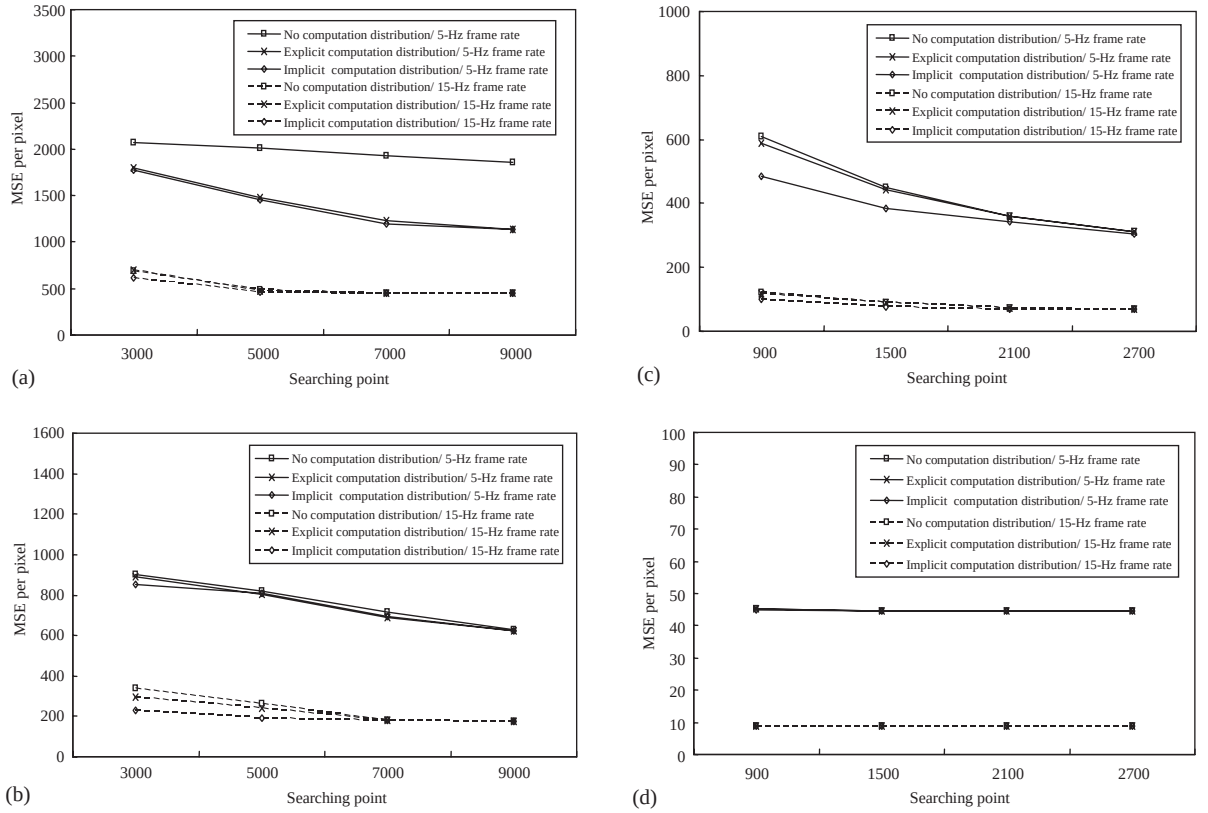

Fig. 4. CD results of MGDS with various computation-distribution schemes.(a) Football sequence. (b) Tennis sequence. (c) Foreman sequence. (d) Salesman sequence.

to 2700 searching points per frame for the QCIF videos. Parameter α for the explicit computation distribution is set to 1.2, and parameters TH_0 and Δ for the implicit computation distribution are fixed at 512 and 256, respectively. Fig. 4 shows the CD results of these algorithms for codecs with the above computational powers. In general, the MGDS with computation-distribution schemes perform better than the MGDS without computation distribution, especially for the football sequence with a 5-Hz frame rate. The figure illustrates the benefit of using computation distribution.

We now analyze the CD performance of the MGDS with explicit and implicit computation distribution. An interesting phenomenon is that the MSE values of MGDS with implicit computation distribution are lower than those of MGDS with explicit computation distribution, especially when the computational power is smaller than 5000 and 1500 searching points per frame for the SIF and QCIF videos, respectively. In this situation, the computation is insufficient such that the computational upper bound of MBs cannot be predicted precisely in the explicit scheme. However, the implicit scheme can efficiently distribute computation since the number of searching points computed is determined by the accuracy of the motion compensation. Thus, MGDS with implicit computation distribution is utilized in the following simulations.

We now discuss the effects of TH_0 and Δ on the implicit computation distribution of MGDS. Because there are two variables to be analyzed, Δ is first fixed to 256. Fig. 5 shows the MSE values of MGDS with TH_0 ranging from 0 to 1536. In

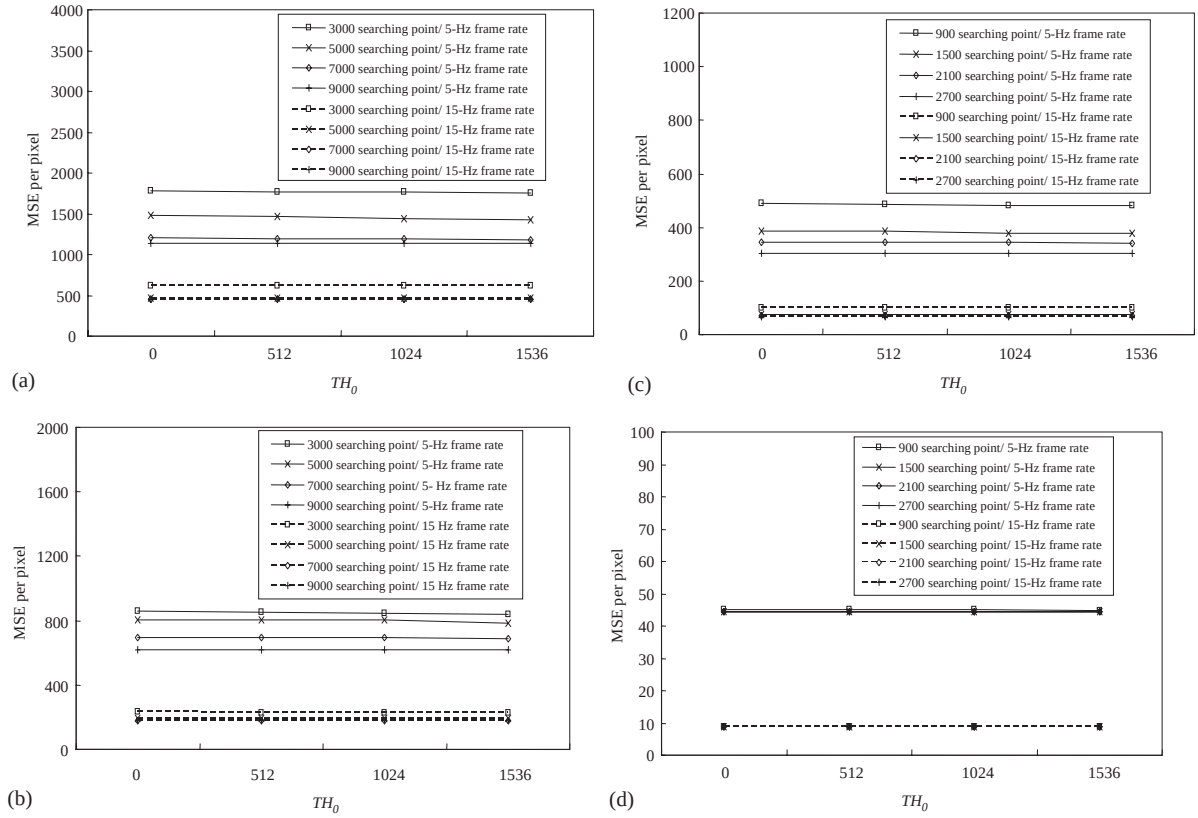

Fig. 5. CD results of implicit computation-distribution scheme with fixed Δ at 256 and different value of TH_0. (a) Football sequence. (b) Tennis sequence. (c) Foreman sequence. (d) Salesman sequence.

over 95% of tested cases the MSE values are almost independent of the value of TH_0 for a constant computational power. This is due to the proposed adaptation mechanism automatically adjusting the stopping threshold to a suitable value dependent on the motion activity of videos and the computational power of the codec. Fig. 6(a) gives an example of the MSE values under various values of TH_0 for the football sequence with a 15-Hz frame rate and a computational power of 3000 searching points per frame. Although the MSE values differ at the beginning of the video, they become equal after the 45th frame. The average MSE values per pixel are 619, 618, 615, and 615 when TH_0 is 0, 512, 1024, and 1536, respectively. Another example is depicted in Fig. 6(b) using the foreman sequence with a 15-Hz frame rate and a computational power of 900 searching points per frame. In this example, MSE values are the same after the sixth frame.

We now discuss the influence of Δ on the implicit computation distribution. Fig. 7 gives the MSE values of MGDS with Δ ranging from 128 to 2048 when TH_0 is set to 512. Similar to the results for TH_0, in over 90% of tested cases the MSE values are almost independent of the value of Δ for a constant computational power. For the football sequence with a 15-Hz frame rate and a computational power of 3000 searching points per frame, the average MSE values per pixel are 621, 618, 623, 629, 628, and 630 when Δ is 128, 256, 512, 1024, 1536, and 2048, respectively. One exceptional case is the tennis sequence with a 5-Hz

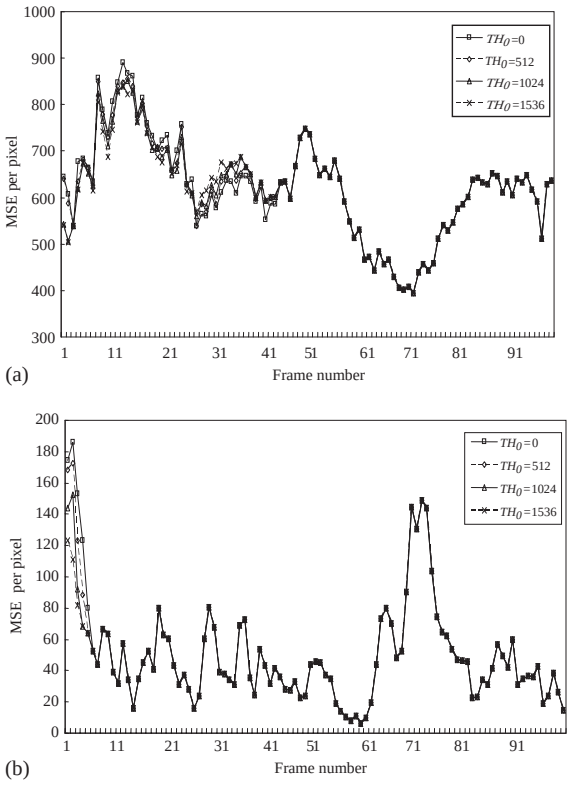

Fig. 6. MSE details of MGDS with fixed Δ at 256 and various values of TH_0. (a) Football sequence with 15-Hz frame rate under 3000 searching points per frame. (b) Foreman sequence with 15-Hz frame rate under 900 searching points per frame.

frame rate and a computational power of 5000 searching points per frame, where the MSE values per pixel are 817, 807, 747, 712, 703, and 724 when Δ is 128, 256, 512, 1024, 1536, and 2048, respectively. In this case a larger Δ achieves better results because a large stopping threshold is required.

The behaviors of GDS and MGDS are analyzed below. The proposed implicit computation-distribution mechanism with $TH_0 = 512$ and $\Delta = 256$ is also used in GDS for comparison. Fig. 8 shows the MSE values of GDS and MGDS under various computational powers. The CD performance of GDS is generally better than that of MGDS, but the difference is small. For the football sequence with a 15-Hz frame rate, the MSE values of GDS are 210, 175, 161, and 158 for 3000, 5000, 7000, and 9000 searching points per frame, respectively, whereas those of MGDS are 227, 192, 179, and 176, respectively. For the salesman sequence with a 15-Hz frame rate, the MSE values for GDS and MGDS are virtually identical (the difference is no greater than 0.5). Although the MSE performance of MGDS would be worse than that of GDS from the viewpoint of computation, in Section 4 we demonstrate that MGDS has the advantage of being easier to implement in hardware. It is interesting that the CD performance of MGDS is better than that of GDS in the football sequence when the frame rate is 5-Hz. This is because the large amount of motion requires enlargement of the search range to extract the corresponding MVs. In fact, GDS is suitable for MVs surrounding the assumed search center. For blocks of rapid motion (especially in the diagonal direction), MGDS performs better because subsequent searches of it are on the basis of a ± 1 search area. Fig. 9 shows a demonstrative example of this phenomenon, where the dotted searching points indicate the saved computation of MGDS compared to GDS.

We now compare the CD performance of MGDS with two other BMAs, traditional FS and TSS, both of which have efficient hardware implementations. The size of the search area in FS and TSS is set to ± 2 and ± 7 to achieve the same computation (25 searching points are examined for every MB). Therefore, the allocated number of searching points per frame in MGDS is set to 8250 and 2475 for SIF and QCIF videos, respectively. Table 2 gives the MSE results for MGDS, TSS, and FS under the specified computation conditions. For the tennis sequence with a 15-Hz frame rate, the MSE value of MGDS is the best because MVs of this sequence approximate a center-biased distribution and MGDS efficiently allocates computation to blocks based on their motion activities. Compared to FS and TSS, MGDS can achieve MSE improvements of 28.07 ($= 205.30 - 177.23$) and 233.99 ($= 411.22 - 177.23$), respectively. For the football sequence with a 15-Hz frame rate, a 135.62 ($= 587.50 - 451.88$) MSE improvement is obtained by MGDS compared to FS. However, the result of MGDS is worse than that of TSS

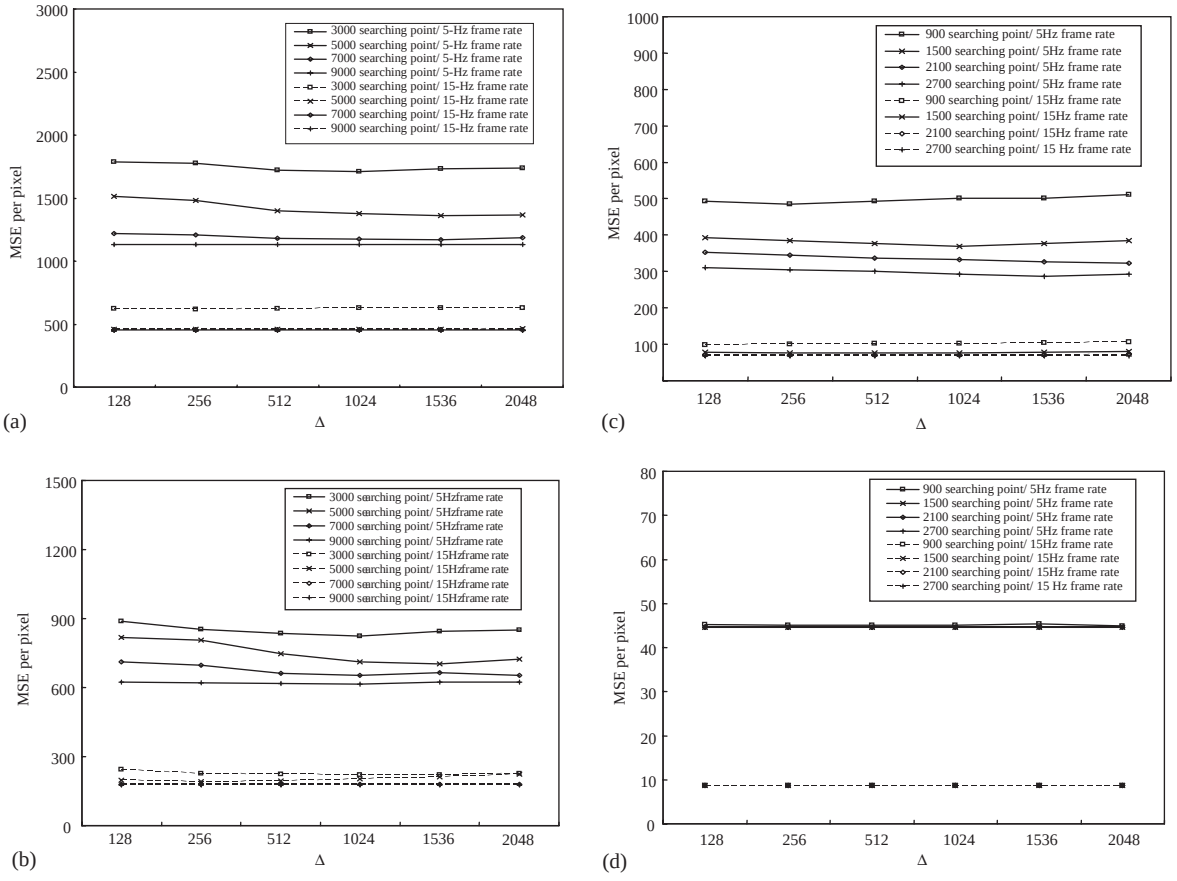

Fig. 7. CD results of implicit computation-distribution scheme with fixed TH_0 at 512 and different value of Δ. (a) Football sequence. (b) Tennis sequence. (c) Foreman sequence. (d) Salesman sequence.

because the football sequence includes large motion activity, for which TSS is more suitable. The MSE degradation between MGDS and TSS is 274.18 ($= 451.88 - 177.70$). For the foreman sequence with a 15-Hz frame rate, the results of MGDS and TSS are better than that of FS. For the salesman sequence with a 15-Hz frame rate, the results of MGDS, TSS, and FS are almost the same due to the very small motion activity in this sequence. This indicates that the CD performance of MGDS is similar to that of TSS. The major advantage of MGDS over TSS is that MGDS can be easily implemented using existing ± 1 FS designs simply. On the other hand, the CD performance of MGDS is better than that of FS. MGDS uniformly achieves a higher quality than FS by an amount that is dependent on motion activities of sequences.

4. MGDS algorithm on full-search array architecture

From the viewpoint of hardware implementation, FS is the best choice for the estimation of block motion. However, the computational complexity of traditional FS is huge because it does not utilize information on motion activity in video sequences. To take advantage of both the design regularity of FS and the information on video motion activity, MGDS is designed such that it can be performed by a series of ± 1 FSs, i.e.,

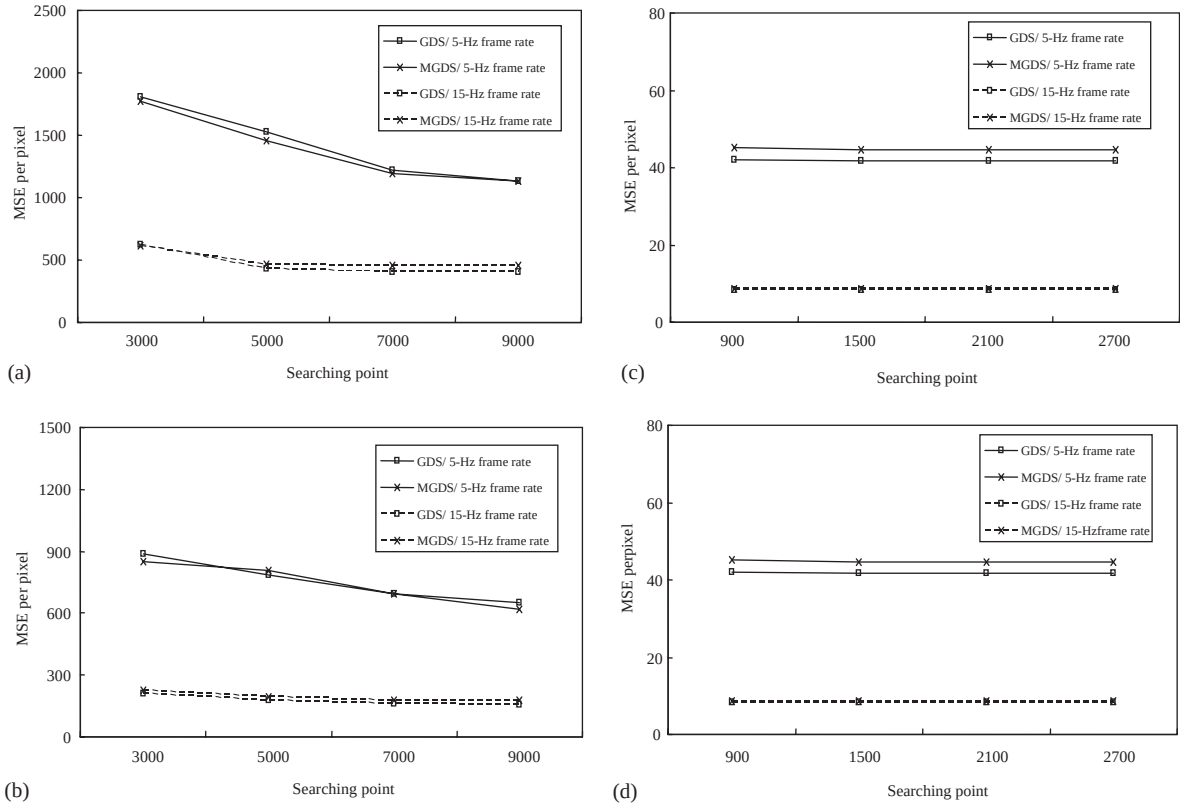

Fig. 8. CD results of GDS and MGDS with $TH_0 = 512$ and $\Delta = 256$. (a) Football sequence. (b) Tennis sequence. (c) Foreman sequence. (d) Salesman sequence.

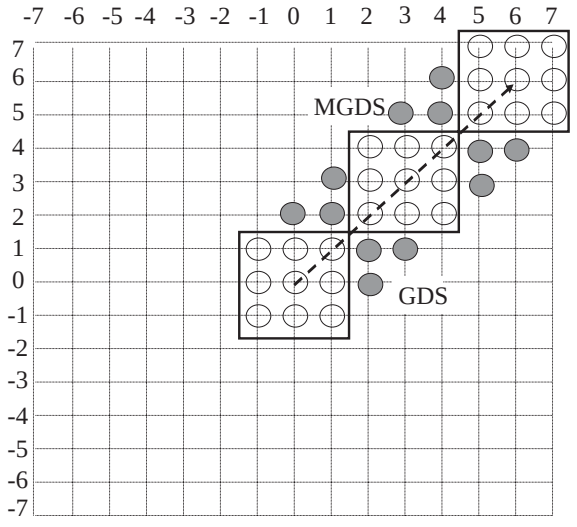

Fig. 9. A diagonal motion example, where dotted searching points are the saved computation of MGDS over GDS.

Table 2
MSE results of MGDS, TSS, and FS under the same computation

	MGDS	TSS	FS
(a) 15 Hz			
Tennis	177.23	411.22	205.30
Football	451.88	177.70	587.50
Foreman	68.01	59.86	90.48
Salesman	8.69	8.42	8.40
(b) 5 Hz			
Tennis	636.01	608.33	843.19
Football	1147.86	1141.01	1680.84
Foreman	317.00	292.33	497.06
Salesman	44.43	41.43	46.47

a square 3×3 search window of nine points. Fig. 10 shows the MGDS structure in a video coding system comprising three modules: memory

module, search engine, and control module. The memory module stores current and referenced frames. The search engine, which represents the kernel of this system, comprises an array of ±1 FSs that work in parallel to extract the best matching points. The number of employed ±1 FSs determines the computational power of the proposed design. The major task of the control module is to control the data flows of MGDS between the memory module and the search engine. Details of the control module and the search engine are addressed below.

4.1. Control module

The threshold-adaptation approach is employed by MGDS to distribute computational power since it can achieve better CD performance. Following Eq. (1), the control module sends the addresses of the current MB X in the current frame and the current search center Y in the reference frame to the DMA (direct memory access) controller of the video coding system. The control module terminates searches of the current MB if one of the stopping conditions is satisfied. The stopping threshold is updated by Eq. (5) on a frame-by-frame basis. The above operations are implemented in software in the video coding system.

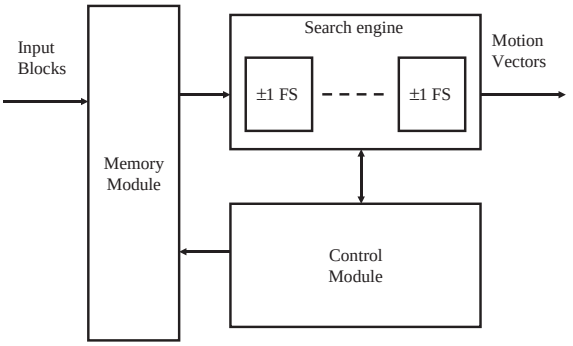

Fig. 10. Structure of the proposed FS array architecture.

4.2. Search engine

Fig. 11 depicts the pixels structure for a ±1 FS, where X and Y represent the pixels of the current MB and the search area, respectively. The MV is computed by searching for the minimum SAD value between X and nine searching points in the searching area. The SAD value is defined as

$$\mathrm{SAD}^{k,l} = \sum_{i=0}^{15}\sum_{j=0}^{15} |X_{i,j} - Y_{(k+i),(l+j)}|, \tag{6}$$

where $-1 \leqslant k \leqslant 1$, $-1 \leqslant l \leqslant 1$, and $X_{i,j}$ and $Y_{i,j}$ are the intensity values of pixels at location (i,j) of X and Y, respectively. To avoid redundant memory accesses, the proposed search engine computes all SAD values of nine searching points concurrently, i.e., $\mathrm{SAD}^{-1,-1}$, $\mathrm{SAD}^{-1,0}$, $\mathrm{SAD}^{-1,1}$, $\mathrm{SAD}^{0,-1}$, $\mathrm{SAD}^{0,0}$, $\mathrm{SAD}^{0,1}$, $\mathrm{SAD}^{1,-1}$, $\mathrm{SAD}^{1,0}$, and $\mathrm{SAD}^{1,1}$. In addition, the distortion in the $\mathrm{SAD}^{k,l}$ is a serial summation of its partial distortions in pipeline

Search Area, Y

(-1, -1)	(-1, 0)	.....	(-1, 15)	(-1, 16)
(0, -1)	(0, 0)	.....	(0, 15)	(0, 16)
. . .	. . .	.....	. . .	. . .
(15, -1)	(15, 0)	.....	(15, 15)	(15, 16)
(16, -1)	(16, 0)	.....	(16, 15)	(16, 16)

Current MB, X

(0, 0)	(0, 1)	.....	(0, 15)
(1, 0)	(1, 1)	.....	(1, 15)
. . .	. . .	.....	. . .
(15, 0)	(15, 1)	.....	(15, 15)

Fig. 11. Pixel structure of the current MB X and search area Y for a ±1 FS.

computing, i.e.,

$$\mathrm{SAD}^{k,l} = \sum_{i=0}^{15} \mathrm{PSAD}_i^{k,l}, \tag{7}$$

where $\mathrm{PSAD}_i^{k,l}$ is the distortion of the ith row in $\mathrm{SAD}^{k,l}$, calculated as

$$\mathrm{PSAD}_i^{k,l} = \sum_{j=0}^{15} |X_{i,j} - Y_{(k+i),(l+j)}|, \quad 0 \leqslant i \leqslant 15. \tag{8}$$

The proposed search engine is implemented as a coprocessor in the video coding system that comprises three units as shown in Fig. 12. The input unit continuously receives pixels of X and Y stored in the memory module row by row from the DMA controller, the PSAD computation unit then evaluates all $\mathrm{PSAD}_i^{k,l}$ values, and finally all $\mathrm{SAD}^{k,l}$ values are serially summed by the SAD summation unit. Below we describe details of the input, PSAD computation, and SAD summation units.

4.2.1. Input unit

The input unit shown in Fig. 13 comprises two parts: the current MB and the search area. For the latter part, three shift registers (R1, R2, and R3) with D-type flip-flops ranging from Rl(-1) to R1(16), R2(-1) to R2(16), and R3(-1) to R3(16) store pixels in the search area. Two pixels of Y are serially shifted into Rl, R2, and R3 per clock cycle, controlled by a 1-3 demultiplexer and a 1-3 counter C1. When C1 is 1, 2, and 3, R1, R2, and R3 are selected, respectively. Since there are 18 pixels in a row of Y, nine clock cycles are required to fetch a row into a shift register. Thus, C1 with an initial value of 1 is cyclically updated every nine clock cycles. For the current MB part, a shift register S1 with 16 D-type flip-flops is used to store pixels of a row in the search area X. We also fetch

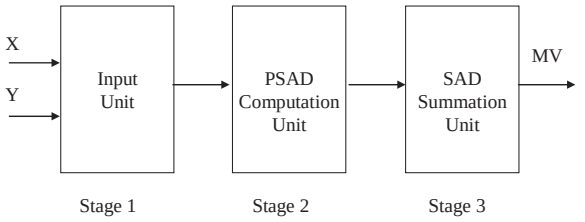

Fig. 12. Structure of the proposed ±1 FS pipeline architecture.

two pixels of X per clock cycle, resulting in eight clock cycles being required. Thus, one idle cycle is necessary to synchronize the data flow between the two parts of the input unit. In addition, an initial delay of 18 clock cycles is required while the search-area part fetches the first and second rows. A timing table of the above operations is provided in Table 3.

The other objective of the input unit is to prepare pixels used for the PSAD computation unit to evaluate $\mathrm{PSAD}_i^{k,l}$. After 27 clock cycles, pixels of the first row in X and the first three rows in Y are ready in shift registers S1, R1, R2, and R3, respectively. Therefore, at the 28th clock cycle the data stored in registers S1 and R1 are simultaneously loaded into registers S and R, respectively. The PSAD computation unit uses these data to compute $\mathrm{PSAD}_0^{-1,-1}$, $\mathrm{PSAD}_0^{-1,0}$, and $\mathrm{PSAD}_0^{-1,1}$ at the 28th, 29th, and 30th clock cycles, respectively. In order to compute the following partial SAD (i.e., $\mathrm{PSAD}_0^{0,-1}$, $\mathrm{PSAD}_0^{0,0}$, $\mathrm{PSAD}_0^{0,1}$), data stored in register R2 are loaded to register R at the 31st clock cycle. Similarly, data stored in register R3 are loaded to register R at the 34th clock cycle to compute $\mathrm{PSAD}_0^{1,-1}$, $\mathrm{PSAD}_0^{1,0}$, and $\mathrm{PSAD}_0^{1,1}$. A total of 16 3-1 multiplexers and a 1-3 counter C2 are used to control the above data flow. C2 is cyclic updated every three clock cycles, and R1(-), R2(-), and R3(-) are simultaneously loaded into register R when C2 is 1, 2, and 3, respectively.

After the above nine clock cycles (i.e., at the 36th clock cycle), pixels of the second row in X are ready in register S1, and the second, third, and fourth row in Y are stored in registers R2, R3, and R1, respectively. During the next nine clock cycles, we arrange data in registers S1, R2, R3, and R1 to compute $\mathrm{PSAD}_1^{-1,-1}$, $\mathrm{PSAD}_1^{-1,0}$, $\mathrm{PSAD}_1^{-1,1}$, $\mathrm{PSAD}_1^{0,-1}$, $\mathrm{PSAD}_1^{0,0}$, $\mathrm{PSAD}_1^{0,1}$, $\mathrm{PSAD}_1^{1,-1}$, $\mathrm{PSAD}_1^{1,0}$, and $\mathrm{PSAD}_1^{1,1}$ in the same manner as described in the above paragraph, except for the loading order of register R. In this period, data must be loaded to R in the order R2, R3, and R1 every three cycles, and this loading order is changed to R3, R1, and R2 at the next nine clock cycles. The loading-order problem of register R can be easily solved by setting the initial value of

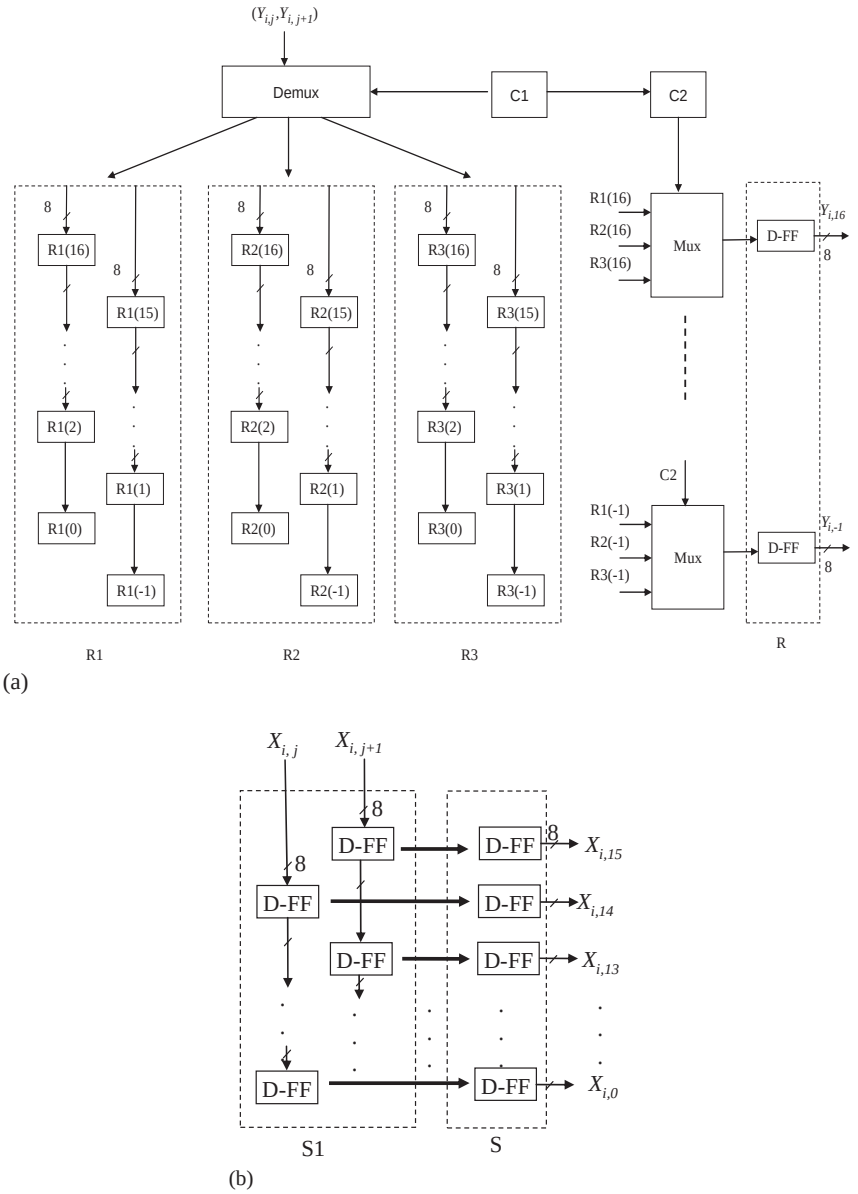

Fig. 13. Block diagram of the input unit. (a) Search area Y, where Mux and Demux represent the multiplexer and demultiplexer, respectively. (b) Current MB X.

counter C2 to the value of counter Cl every nine clock cycles.

4.2.2. PSAD computation unit

The PSAD computation unit computes $\mathrm{PSAD}_i^{k,l}$, i.e., the partial $\mathrm{SAD}^{k,l}$ of the ith row. Fig. 14 shows the block diagram of this unit, which comprises two subunits: absolute-difference and adder-tree subunits. The intensity values of pixels from $X_{i,0}$ to $X_{i,15}$ and from $Y_{k+i,l+0}$ to $Y_{k+i,l+15}$ are simultaneously loaded into the absolute-difference subunit comprising 16 subtraction and

Table 3
Timing table for the proposed ±1 FS design

T	Input unit		PSAD computation unit	SAD summation unit
1	$Y_{-1,-1}, Y_{-1,0}$			
⋮		⋮		
9	$Y_{-1,15}, Y_{-1,16}$			
10	$Y_{0,-1}, Y_{0,0}$			
⋮		⋮		
18	$Y_{0,15}, Y_{0,16}$			
19	$Y_{1,-1}, Y_{1,0}$			
20	$Y_{1,1}, Y_{1,2}$	$X_{0,0}, X_{0,1}$		
⋮		⋮		
27	$Y_{1,15}, Y_{1,16}$	$X_{0,14}, X_{0,15}$		
28	$Y_{2,-1}, Y_{2,0}$		$\mathrm{PSAD}_0^{-1,-1}$	
29	$Y_{2,1}, Y_{2,2}$	$X_{1,0}, X_{1,1}$	$\mathrm{PSAD}_0^{-1,0}$	$\sum_{i=0}^{0} \mathrm{PSAD}_i^{-1,-1}$
⋮	⋮	⋮	⋮	⋮
36	$Y_{2,15}, Y_{2,16}$	$X_{1,14}, X_{1,15}$	$\mathrm{PSAD}_0^{1,1}$	$\sum_{i=0}^{0} \mathrm{PSAD}_i^{1,0}$
37	$Y_{3,-1}, Y_{3,0}$		$\mathrm{PSAD}_1^{-1,-1}$	$\sum_{i=0}^{0} \mathrm{PSAD}_i^{1,1}$
38	$Y_{3,1}, Y_{3,2}$	$X_{2,0}, X_{2,1}$	$\mathrm{PSAD}_1^{-1,0}$	$\sum_{i=0}^{1} \mathrm{PSAD}_i^{-1,-1}$
⋮	⋮	⋮	⋮	⋮
45	$Y_{3,15}, Y_{3,16}$	$X_{2,14}, X_{2,15}$	$\mathrm{PSAD}_1^{1,1}$	$\sum_{i=0}^{1} \mathrm{PSAD}_i^{1,0}$
⋮		⋮	⋮	
171			$\mathrm{PSAD}_{15}^{1,1}$	$\mathrm{SAD}^{1,0} = \sum_{i=0}^{15} \mathrm{PSAD}_i^{1,0}$
172				$\mathrm{SAD}^{1,1} = \sum_{i=0}^{15} \mathrm{PSAD}_i^{1,1}$
				Output MV

absolute-value circuits that simultaneously evaluate

$$\mathrm{AD}_{i,j}^{k,l} = |X_{i,j} - Y_{(k+i)(l+j)}|, \quad 0 \leqslant j \leqslant 15. \tag{9}$$

The adder-tree subunit adds the results from $\mathrm{AD}_{i,0}^{k,l}$ to $\mathrm{AD}_{i,15}^{k,l}$ and outputs $\mathrm{PSAD}_i^{k,l}$. The adder tree comprises three levels of 4-2 adders whose output format is the carry-saved form in order to decrease the propagation delay during the accumulation operations. Finally, a carry look-ahead adder is adopted to translate the carry-saved form into the binary representation.

4.2.3. SAD summation unit

This unit evaluates SAD values of the nine searching points of a ±1 FS and outputs the MV. The structure of the SAD summation unit is depicted in Fig. 15, where a cyclic shift register with nine D-type flip-flops stores SAD values, an adder accumulates partial SAD values using Eq. (7), and a comparator maintains the minimum SAD. The input of this unit is the output of the PSAD computation unit, which is in the following order: $\mathrm{PSAD}_i^{-1,-1}$, $\mathrm{PSAD}_i^{-1,0}$, $\mathrm{PSAD}_i^{-1,1}$, $\mathrm{PSAD}_i^{0,-1}$, $\mathrm{PSAD}_i^{0,0}$, $\mathrm{PSAD}_i^{0,1}$, $\mathrm{PSAD}_i^{1,-1}$, $\mathrm{PSAD}_i^{1,0}$, and $\mathrm{PSAD}_i^{1,1}$, where i is from

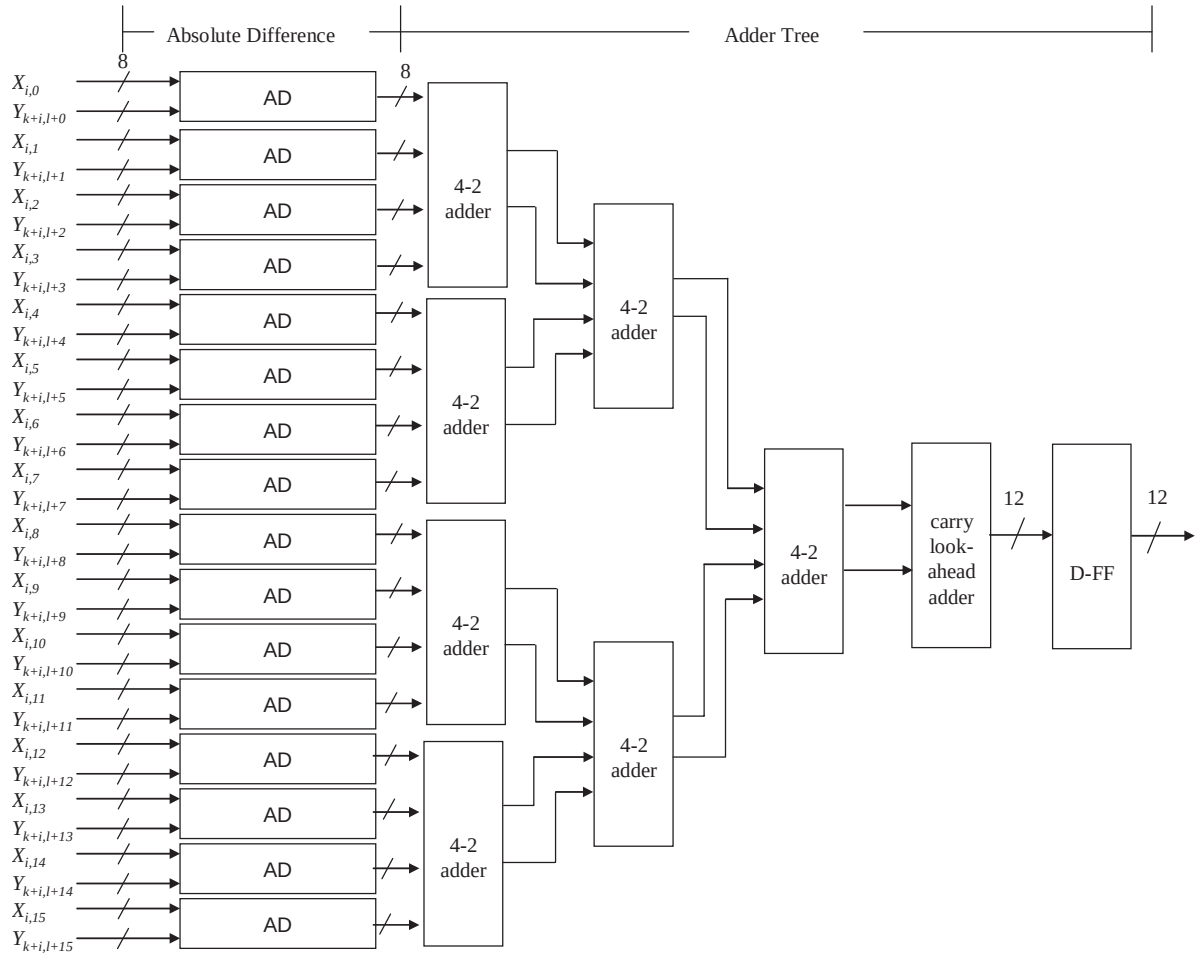

Fig. 14. Block diagram of the PSAD computation unit.

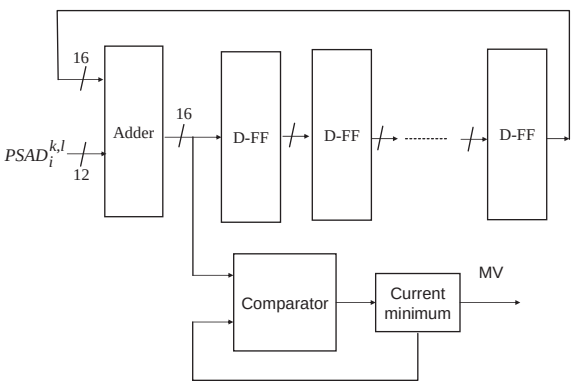

Fig. 15. Block diagram of the SAD summation unit.

0 to 15. Once a partial SAD is generated, it is accumulated in the leftmost flip-flop in the shift register by the adder and the result is then cyclically shifted for the next SAD. The processing of all partial SAD values results in the serial generation of $SAD^{-1,-1}$, $SAD^{-1,0}$, $SAD^{-1,1}$, $SAD^{0,-1}$, $SAD^{0,0}$, $SAD^{0,1}$, $SAD^{1,-1}$, $SAD^{1,0}$, and $SAD^{1,1}$, and hence only one comparator is required to obtain the minimum SAD.

4.2.4. Layout

The engine shown in Fig. 16 is implemented by the UMC 0.18-um Artisan Cell library. There are 3770 cells and 52 IO pads in this design, the area of

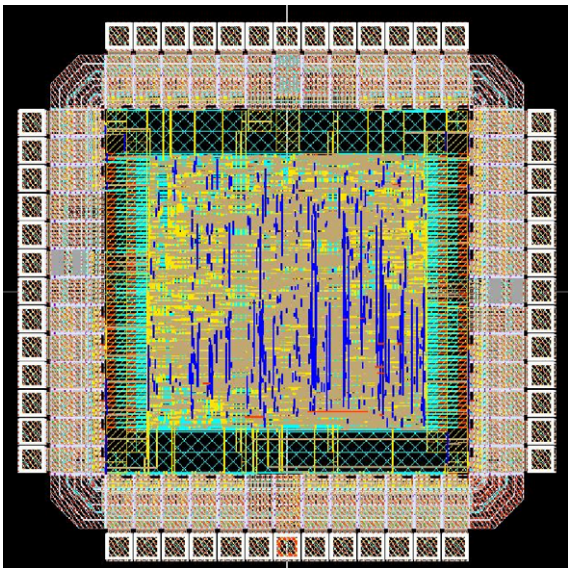

Fig. 16. Layout of the search engine.

which is $1200 \times 1200\,\mu m^2$. The core area is $600 \times 600\,\mu m^2$, and operates at 166 MHz and 1.8 V. In this architecture, 172 clock cycles are needed to perform a ± 1 FS, including the latency of initialization. Therefore, the total time required for a ± 1 FS is 1032 ns. Note that the proposed design is only one example of the search engine which can also be formed by other existing hardware designs of FS.

5. Conclusions

FS is the best choice for block motion estimation from the viewpoint of hardware implementation, but the computational complexity of traditional FS is huge. This paper proposes a simple architecture with a ± 1 FS array and an efficient algorithm named MGDS to utilize the advantage of FS whilst also overcoming its drawbacks. The basic component of ± 1 FS array can be constructed using efficient hardware designs of FS. In addition, the corresponding computational power can be easily updated by changing the number of elements in the FS array and efficiently distributed to blocks or frames of video sequences by a stopping threshold adaptation mechanism. Our resulting architectural and algorithmic codesign for block motion estimation is both simple and efficient.

References

[1] Chein-Wei Jen, Jen-Chieh Tuan, Tian-Sheuan Chang, On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture, IEEE Trans. Circuits Systems Video Technol. 12 (1) (January 2002) 61–72.

[2] Chun-Ho Cheung, Lai-Man Po, A novel cross-diamond search algorithm for fast block motion estimation, IEEE Trans. Circuits Systems Video Technol. 12 (12) (December 2002) 1168–1177.

[3] H.-M. Hang, Y.-M. Chou, S.-Chih.Cheng, Motion estimation for video coding standards, J. VLSI Signal Process. Systems for Signal, Image, Video Technol. (November 1997) 113–136.

[4] Information Technology—Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s: Video, ISO/IEC 11 172-2 (MPEG-1 Video), 1993.

[5] Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video, ISO/IEC 13 818-2-ITU-T Rec. H.262 (MPEG-2 Video), 1995.

[6] Information Technology—Generic Coding of Audio-Visual Objects Part 2: Visual, ISO/IEC 14 496-2 (MPEG-4 Video), 1999.

[7] T. Koga, K. Iinuma, A. Iijima, T. Ishiguro, Motion-compensated interframe coding for video conderencing, Proc. NTC81. New Orleans, LA, 1981, pp. C9.6.1–9.6.5.

[8] T. Komarek, P. Pirsch, Array architecture for block matching algorithms, IEEE Trans. Circuits Systems (October 1989) 269–277.

[9] R. Li, B. Zeng, M.L. Liou, A new three-step search algorithm for block motion estimation, IEEE Trans. Circuits Systems Video Technol. 4 (August 1994) 438–442.

[10] P. Pirsch, N. Demassieux, W. Gehrke, VLSI architecture for video compression—a survey, Proc. IEEE (February 1995) 220–246.

[11] L.M. Po, W.C. Ma, A novel four-step search algorithm for fast block motion estimation, IEEE Trans. Circuits Systems Video Technol. 6 (3) (1996) 313–317.

[12] Shih-Chang Hsia, VLSI implementation for low-complexity full-search motion estimation, IEEE Trans. Circuits Systems Video Technol. 12 (7) (July 2002) 613–619.

[13] Standardization Sector of ITU, Video Coding for Low Bitrate Communication, ITU-T Rec. H.263, March 1996.

[14] A.M. Tourapis, O.C. Au, M.L. Liou, Predictive motion vector field adaptive search technique (PMVFAST)—enhancing block based motion estimation, Proceedings of

Visual Communications and Image Processing 2001 (VCIP-2001), San Jose, CA, January 2001.

[15] C.de Vleeschouwer, T. Nilsson, K. Denolf, J. Bormans, Algorithmic and architectural co-design of a motion-estimation engine for low-power video devices, IEEE Trans. Circuits Systems Video Technol. 12 (12) (December 2002) 1093–1105.

[16] S. Zhu, K.-K. Ma, A new diamond search algorithm for fast block-matching motion estimation, IEEE Trans. Image Processing 9 (February 2000) 287–290.