Neural, Parallel & Scientific Computations 15 (2007) 45-58

Parallel Motion Estimation on the MDSP Multiprocessor

Yen-Chun Lin and Chuan-Da Hsiung
Dept. of Computer Science and Information Engineering
National Taiwan University of Science and Technology
Taipei 106, Taiwan

Abstract

Motion estimation (ME) is a process for removing temporal redundancies in video sequences. It contributes most of the compression ratio and consumes typically 60-80% of the video encoding time. This paper investigates parallel execution of ME on the Multiprocessor DSP (MDSP) architecture, a software programmable, scalable multiprocessor platform. We use a low-overhead data distribution scheme and achieve load balancing among processors. In addition, we assign different tasks of an ME algorithm to suitable types of processors. Many sequential ME algorithms can be adapted to run on the MDSP in parallel, and the experimental results show that very good speedup can be obtained. Since the MDSP is software programmable, our design can be easily integrated into a video encoder, modified to meet different standards, and used in various video applications.

Keywords - motion estimation, multiprocessor, parallel execution, speedup, video encoding

1. INTRODUCTION

Motion estimation (ME) is a process for removing temporal redundancies in video sequences (Shi & Sun, 1999). It is based on the assumption that the changes between successive frames come from motion of objects in the video. The amount of data can be reduced through coding the displacements of objects. ME is central to video compression. It contributes most of the compression ratio and consumes typically 60-80% of the video encoding time (Kuhn, 1999; Zhu et al., 2002). That is, ME has a great impact on the speed of a video encoder, and is important for real-time video applications, such as video conferencing, and digital TV broadcasting.

Received August 15, 2005; final form December 11, 2007 1061-5369 $15.00

A video encoder can be implemented in hardware or software. Hardware implementations use customized architectures. They are usually optimized for specific algorithms, thus are very fast. However, they are inflexible, unsuitable for other digital video applications or future standards (Ahmad et al., 2001). The hardware approach also has higher technical hurdles and needs a longer design cycle (Bolton et al., 2002). ASIC motion estimators have been designed (Fanucci et al., 2001; Kittitornkun & Hu, 2001; Kuhn, 1999; Parhi & Nishitani, 1999; Tuan et al., 2002), but fast ME algorithms are generally hard to realize in hardware for their irregular control flow and data flow.

The software approach does not have the above-mentioned drawbacks. However, the intensive computation of video compression may overwhelm a single-processor computer. Thus, parallel processing becomes a natural choice. Vos and Schobinger design a two-dimensional systolic array to execute multiple ME algorithms (Vos & Schobinger, 1993). To achieve the same objective, Dutta and Wolf propose a parallel architecture (Dutta & Wolf, 1996). Their design avoids the drawbacks of memory-access bottlenecks from which general-purpose multiprocessors suffer; PEs are connected to memory banks through a general-purpose interconnection network. Dutta and Wolf also survey various parallel approaches to ME. Tan et al. implement full-search ME algorithms on four distributed-memory parallel computers (Tan et al., 1999). Their results should help practitioners in using parallel systems. Some other researchers use parallel computers (Ahmad et al., 2001; Akramullah et al., 1995; Shen & Delp, 1996) or clusters of workstations (Akramullah et al., 1999; He et al., 1999) as the computing platform; their hardware costs are quite high. Kang et al propose a special hardware/software co-design architecture (Kang et al., 2003); they specifically design a data distribution mechanism to use the tree-structured processor-memory nodes efficiently.

This paper investigates parallel execution of ME on the Multiprocessor DSP (MDSP) architecture, a software programmable, scalable multiprocessor platform (Cradle, 2004; Wyland, 1999). The MDSP is a special shared-memory multiprocessor with programmable I/O. It is fast enough for video encoding, and much less expensive than the above-mentioned parallel hardware. It employs multiple RISC-like processors, digital signal processors, and direct memory access (DMA) units to provide a high-performance parallel processing platform. More details of the MDSP architecture will be described in a later section.

We use a low-overhead data distribution scheme and achieve load balancing among processors. In addition, we assign different tasks of an ME algorithm to suitable types of processors. Many sequential ME algorithms can be adapted to run on the MDSP in parallel, and the experimental results show that very good speedup can be obtained. Since the MDSP is software programmable, our design can be easily integrated into a video encoder, modified to meet different standards, and used in various video applications.

The rest of this paper is organized as follows. Sec. 2 briefly introduces sequential

ME techniques. Sec. 3 describes the MDSP architecture. Sec. 4 uses a fast ME algorithm to demonstrate how sequential ME algorithms can be adapted to fully utilize the MDSP architecture and resources. Sec. 5 gives experimental results and some related discussions. Finally, Sec. 6 concludes this paper.

2. BRIEF INTRODUCTION TO SEQUENTIAL ME TECHNIQUES

The block matching technique has been the most widely used for ME, and has been adopted by international video coding standards, such as MPEG and H.26x series (1995; 1999; 1993; 1996). It divides a frame into non-overlapped blocks of $B \times B$ pixels. The most frequently used B is 16; a block of size 16×16 is called a macroblock. For each current macroblock, a best matched counterpart is found within the search window in its reference frame. The reference frame is another frame reconstructed from encoded data. The displacement between the position of the best match and the position of the current macroblock is called motion vector (MV). It is recorded together with the prediction error matrix (PEM), which is a 16×16 matrix of pixel value differences between the current macroblock and the best match. Thus, the decoder can reconstruct the macroblock by using the MV, for locating a macroblock from the reference frame, and the PEM.

Of all the matching criteria, the sum of absolute difference (SAD), also called the mean absolute difference (MAD) (Koga et al., 1981), is the most widely used owing to its simplicity and effectiveness. SAD is defined as

$$SAD_{(x,y)}(u,v) \equiv \sum_{j=0}^{B-1}\sum_{i=0}^{B-1}\left|I_t(x+i,y+j)-I_{t-1}(x+u+i,y+v+j)\right|, \tag{1}$$

where $I_t(m, n)$ and $I_{t-1}(m, n)$ respectively stand for the intensity value of pixels at (m, n) in the current and the reference frame. Equation (1) calculates the SAD value between the macroblock at (x, y) in the current frame and a candidate macroblock for the best match at $(x + u, y + v)$ in the reference frame. Every valid pair of (u, v) stands for a possible candidate for the best match. ME is to find a pair of (u, v) that minimizes Equation (1).

The most intuitive ME algorithm is full search. Full search is a brute force algorithm that exhaustively checks every candidate macroblock within the search window. It assures that the globally best match can be found; however, since the SAD calculation is nontrivial, full search needs plenty of calculations.

Therefore, many fast ME algorithms, or fast searches, have been proposed. Most of them are based on the unimodel error surface assumption; that is, for candidate macroblocks, the closer to the global best match, the less the SAD value. These algorithms have their own search patterns, each of which defines a small set of search points. Each search point stands for a candidate macroblock. Instead of examining every

search point in the search window that full search does, fast searches only examine the search points on the search pattern to conjecture the possible location of the best match, and then move the search pattern toward it. Examples of this kind of algorithms include three-step search (TSS) (Koga et al., 1981), new three-step search (Li et al., 1994), four-step search (Po & Ma, 1996), diamond search (DS) (Zhu & Ma, 2000), PMVFAST (Tourapis et al., 2000), hexagonal search (Zhu et al., 2002), cross-diamond search (Cheung & Po, 2002), and cross-diamond-hexagonal search (Cheung & Po, 2005). They greatly reduce encoder's computational workload at the expense of both picture quality and compression ratio, because the unimodel error surface assumption does not always hold and the search process may thus trap into local minima.

The correlation among adjacent macroblocks can be exploited to help some fast searches. The MV of the current macroblock is usually the same as or similar to the MVs of temporally or spatially neighboring macroblocks, which can be used to predict the MV of the current macroblock (Alkanhal et al., 1999; Nam et al., 2000; Tourapis et al., 2000; Xu et al., 1997).

The MVs used for predicting the MV of the current macroblock are called predictors. The most frequently used predictors include the MVs of the left, above, and above-right macroblocks of the current macroblock. MV prediction usually helps fast searches reach the best match earlier and avoids trapping into local minima.

3. MDSP MULTIPROCESSOR

The MDSP architecture is shown in Figure 1. An MDSP chip contains at least one processor subsystem, which is a cluster of processors. Processor subsystems are connected to a very high-speed global bus, and communicate with the external DRAM through a DRAM controller. The MDSP has a programmable I/O system, which enables developers to implement most of the I/O devices in software. The programmable I/O system consists of a special I/O subsystem, two I/O buses and logic at the I/O pins; this hardware is powerful enough to implement high performance devices such as PCI, Ethernet, and SCSI interfaces. The synchronization mechanism in the MDSP is provided by 64 global semaphore registers.

Figure 2 shows the processor subsystem block diagram. A processor subsystem has four Media Stream Processors (MSPs), each of which consists of a Processing Engine (PE) and two Digital Signal Processors (DSPs). In addition, the processor subsystem has 32 KB program memory shared by PEs and 64 KB data memory shared by all the processors in the processor subsystem. The processor subsystem also has a DMA controller called Memory Transfer Engine (MTE), which can be used to transfer data between the shared memories and external DRAM. The MTE has four program counters, thus can deal with four data transfer requests simultaneously. All the processors in the

processor subsystem are connected together with the shared memories through a local bus. There are 32 local semaphore registers in the processor subsystem.

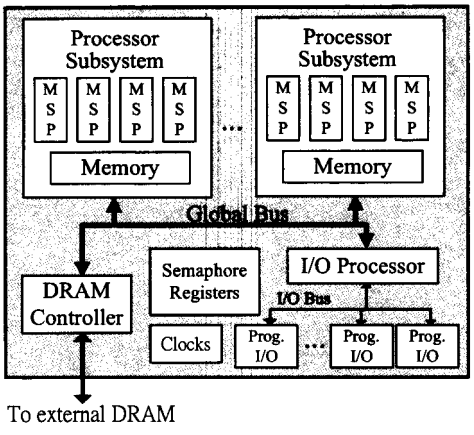


Fig. 1. MDSP block diagram.

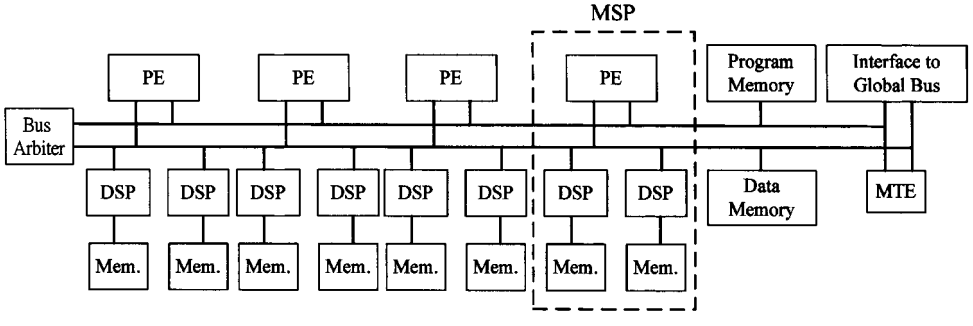


Fig. 2. Processor subsystem block diagram.

The PE is a 32-bit RISC processor and operates at about 45 MIPS under 225 MHz. It can initiate and monitor the MTE and the two collaborative DSPs, thus is ideally suited for control functions.

A DSP is a 32-bit processor; it is the primary computing engine of the MDSP. It operates at 220 MHz, and has 128 registers and a local program memory of 512 20-bit instructions. In addition, it has a floating point multiplier-accumulator that can perform sixteen 8-bit operations, four 16-bit operations, or three floating point operations in each clock cycle; thus, it can provide up to 7.04 GOPS. The DSP also has two 16-word FIFOs that allow data pre-fetch from and post-write to the processor subsystem shared data memory while executing instructions, maintaining a high processing rate with efficient memory transfers.

4. ADAPTING SEQUENTIAL ME ALGORITHMS TO THE MDSP

4.1. Balancing the workload of MSPs

Every macroblock in the current frame is processed separately. Each MSP can be assigned a macroblock to perform ME. The single-program-multiple-data approach is used. An MSP acts as the controller. It keeps an integer variable, say *next_MB*, to indicate the next macroblock to be processed. Every MSP consults *next_MB* to decide the next macroblock for it to perform ME. After an MSP has processed a macroblock, the MSP saves the MV and PEM in the external DRAM before it consults *next_MB* for performing one more ME. Since the MDSP has built-in semaphores, it is easy to ensure the integrity of the read-and-increment operations performed by MSPs on *next_MB*.

For example, suppose the sequence number of macroblocks in a frame begins with 0, the initial value of *next_MB* is 0. When an MSP reads the value of *next_MB*, it also increments *next_MB*. When the value of *next_MB* equals the total number of macroblocks in a frame, it means that all the macroblocks in the current frame have been taken care of. MSPs may be idle for a period of time less than the time required to compute the MV of a macroblock. Therefore, the workload of MSPs is well balanced.

4.2. Task assignment in an MSP

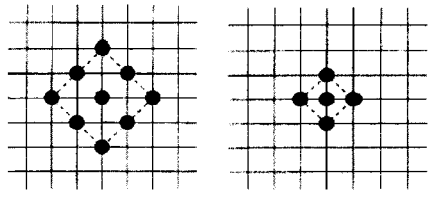
We now consider how the processors of an MSP perform fast search. Recall that many fast searches have their respective search patterns, and every search point of the search pattern represents a candidate macroblock. Fast searches compute the SAD of each search point to approach the possible location of the best match.

We use the DS algorithm (Zhu & Ma, 2000) to illustrate how a fast search can be implemented. DS has two search patterns, i.e., large diamond search pattern (LDSP) and small diamond search pattern (SDSP), depicted in Fig. 3. DS works as follows:

(1) The initial LDSP is at the center of the search window, and the SAD values of the 9 search points are computed. If the point with the smallest SAD is at the center, go to (3); otherwise, go to (2).

(2) The point with the smallest SAD becomes the center of a new LDSP. If the point with the smallest SAD is at the center of the new LDSP, go to (3); otherwise, repeat this step.

(3) The point with the smallest SAD becomes the center of an SDSP. Compute the SAD values of the five search points of the SDSP. The point with the smallest SAD is what we want for computing the MV.



(a) (b)

Fig. 3. Two search patterns of DS: (a) LDSP, (b) SDSP.

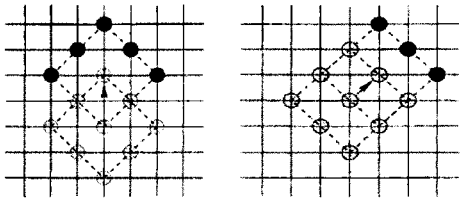
Computing the SAD of a search point is independent of computing that of another point. Therefore, these computations can be done in parallel. Each MSP has two DSPs to compute the SAD values for search points in parallel. As already mentioned, the DSP is fast; in particular, each DSP can perform multiple operations in each clock cycle.

On the other hand, the PE executes the main program of the fast search. It chooses the search point that needs to compute SAD values based on the search pattern and computed SAD values, and asks DSPs to compute new SAD values. DSPs save the SAD values in the shared data memory, from where the PE can read out. If a different ME algorithm is used, only the portion of program that runs on the PE needs rewriting.

4.3. Practical techniques to improve performance

When only one processor subsystem is used, the *next_MB* variable mentioned above should be kept in the 64-KB shared data memory. However, when the system has more processor subsystems, *next_MB* should be stored in the external DRAM. Storing in the shared data memory of one processor subsystem requires complex programming for MSPs of other processor subsystems to access the variable through the global bus (see Fig. 1) and incurs overhead. Storing in the external DRAM is easier to implement and, as will be seen in Sec. 5, does not have perceivable adverse effect on the speedup.

Duplicate computations can be avoided by saving computed SAD values of search points in the shared data memory. For example, Fig. 4 shows two possible states of DS. If the LDSP moves up, as shown in Fig. 4(a), only five new search points need to compute SAD values. If the LDSP moves northeast, as Fig. 4(b) shows, only three points need further computations. We can use a 2-D array to save the SAD values of all the search points. Alternatively, hashing can be used to reduce the amount of memory required.



(a) (b)

Fig. 4. (a) The LDSP moves upward; (b) the LDSP moves northeast.

Different macroblocks may have many pixels in common. Fig. 5 shows that two candidate macroblocks and their overlapping. Therefore, it can save much time to read the whole search window into the shared data memory once and for all. Suppose the search window is [-7, 7], the whole search window of a macroblock takes up $(7 \times 2 + 16)^2 = 900$ bytes of memory, and four MSPs totally require $900 \times 4 = 3600$ bytes of shared memory.

Furthermore, neighboring macroblocks in the current frame have common pixels, as shown in Fig. 6(b). Thus, reading the search windows of multiple macroblocks to the shared data memory at a time can save I/O time.

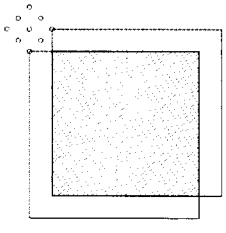
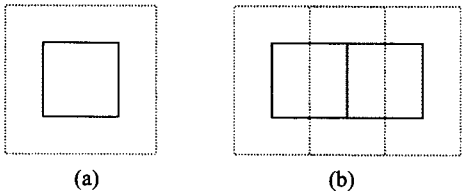


Fig. 5. Overlapped pixels of two candidate macroblocks.



(a) (b)

Fig. 6. (a) A macroblock and its search window;
(b) two search windows with shared pixels.

4.4. Suitability of ME algorithms

Recall that we use PEs and DSPs to do different tasks that can fully utilize their respective capability. Since PEs choose the search points and ask DSPs to compute SAD values, our approach is suited to all block-matching ME algorithms that are based on comparing SAD values, to which full search and most fast searches belong.

It should be noted that the MV prediction mentioned in Sec. 2 needs modification. The usual predictors include the MVs of the above, above-right, and left macroblocks of the current macroblock. However, in our approach, when a macroblock is being processed by an MSP, the macroblock to its left is very likely being processed by another MSP; thus, the MV of the left macroblock is not computed early enough for it to be a predictor. The MV of the above-left macroblock can be used instead.

5. EXPERIMENTAL RESULTS

A programming language CLASM (C-Like Assembly) can be used for software development on the MDSP (Cradle, 2004). Developers can also use the standard ANSI C to code the program executing on PEs. The MDSP has a simulator called Inspector under the Microsoft Windows environment.

To evaluate the effectiveness of our design, we have conducted experiments with different numbers of MSPs. DS is adapted to run on MSPs, and the search window is [-7, 7]. Programs are executed under the Inspector to count the number of MDSP clock cycles required. Source video frames are processed one by one. Every frame uses the uncompressed frame immediately before it as its reference frame.

The controller MSP is in charge of data I/O. To achieve this, the controller MSP will initially load two successive frames, i.e., a current frame and its reference frame, to the external DRAM. Then, when a frame is being processed, all the MSPs, with the possible exception of the controller MSP, perform ME for the macroblocks in the frame, and transfer the results to the external DRAM. In the meantime, the controller MSP can output the results of the previous frame from the external DRAM to the hard disk; this is referred to as *output*. The controller MSP also reads the next frame from the hard disk into memory; this is referred to as *input*. There is no output when the first frame is being processed, and no input when the last frame is being processed. Once the output and input are completed, the controller MSP can perform ME. Therefore, the processing of each frame, except for the first and last frames, by the MSPs can be conceptually illustrated by Fig. 7.

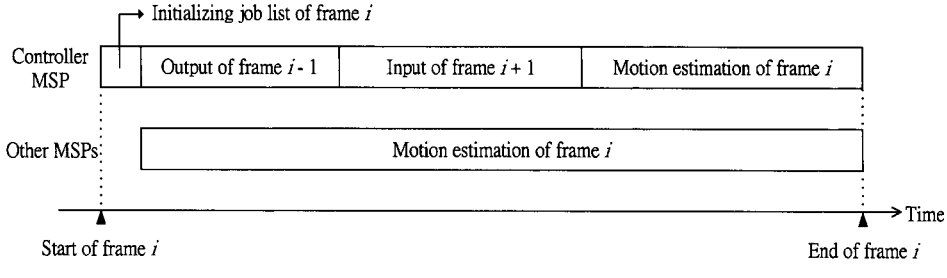


Fig. 7. Processing of a regular frame by MSPs.

As already mentioned, we used uncompressed frames as reference frames. This is different from the case used in real ME, in which a reference frame is actually decompressed after compression. Because the two kinds of reference frame are very similar, it is assumed that and they make little difference to the movement of search patterns in the search process. As a result, the processing time and speedup will only be slightly affected, if any. The speedup measured should be accountable.

In our experiment, before the ME of each macroblock begins, PE reads the whole search window in the reference frame into the shared data memory. As mentioned in Sec. 4.3, this avoids reading the same pixels for different candidate macroblocks. The search window is a square with $16 + 7 \times 2 = 30$ pixels on each side, where each row of 30 pixels takes up 30 bytes of consecutive data. We did not use the MTE to read in the whole search window. If the whole search window is read by the MTE, the MTE will be used 30 times, each for only 30 bytes. This should be ineffective.

We used the first 20 frames of source video sequences of Coastguard, Foreman, and Salesman, each with QCIF (176×144) format. For comparison, we also experiment with the first 50 frames of Salesman. Table I gives the speedups obtained when different numbers of MSPs were used, where Salesman_1 and Salesman_2 represent the first 20 frames and the first 50 frames of Salesman, respectively. When n MSPs were used, the speedup is computed by the number of clock cycles when an MSP is used divided by the number of clock cycles when n MSPs are used.

Table 1. Speedups obtained when different numbers of MSPs were used.

No. of MSPs	Coastguard	Foreman	Salesman_1	Salesman_2
2	1.97	1.97	1.97	1.98
3	2.91	2.91	2.91	2.95
4	3.81	3.82	3.81	3.88
6	5.41	5.47	5.42	5.60
8	7.01	7.09	7.01	7.32
12	7.89	8.61	7.91	8.16

Table 1 shows that when no more than 8 MSPs were used, the speedups were good; however, when 12 MSPs were used, the speedup were not satisfactory. To investigate the cause, we recorded which MSP processed which macroblock. It was found that when more than 8 MSPs were used, virtually no macroblocks of any frame, except the first and last frames, were processed by the controller MSP. This implies that when more than 8 MSPs were used, the ME computation time was not greater than the I/O time. Only when the first frame was being processed, the controller MSP performing no output, and when

the last frame was being processed, the controller MSP performing no input, could the controller MSP help perform ME. Thus, the speedup could only be slightly increased when more than 8 MSPs were used.

Although the I/O in our experiment limits the growth of speedup, the real motion estimator implemented using the MDSP may not perform I/O. Before ME begins, video sequences could have been read to the external DRAM through a specific input device; moreover, the result of ME should also be kept in the external DRAM for further compression, e.g., intracoding. The controller MSP can perform ME after simply setting *next_MB*. Therefore, more MSPs can be used to obtain better speedup.

6. CONCLUSION

We have investigated full utilization of the multiple processors and other resources in the MDSP to perform ME in parallel. Many fast sequential ME algorithms can be adapted to run concurrently on the MDSP to speed up the video compression. We can share the workload among multiple MSPs of the MDSP with minimal overhead, and the MSPs are essentially equally loaded. Our experimental results show that the speedup is very good when no more than 8 MSPs are used. In fact, in real applications, more MSPs can be used to run faster and achieve satisfactory speedup.

REFERENCES

1. Ahmad, I., Akramullah, S. M., Liou, M. L., & Kafil, M. (2001). A scalable off-line MPEG-2 video encoding scheme using a multiprocessor system. Parallel Computing, v. 27, pp. 823-846.
2. Akramullah, S. M., Ahmad, I., & Liou, M. L. (1995). A data-parallel approach for real-time MPEG-2 video encoding. Journal of Parallel and Distributed Computing, v. 30, pp. 129-146.
3. Akramullah, S. M., Ahmad, I., & Liou, M. L. (1999). Parallel MPEG-2 encoder on ATM and Ethernet-connected workstations. Proc. 1999 Int. Conf. of the Austrian Center for Parallel Computation, Salzburg, Austria, pp. 572-574.
4. Alkanhal, M., Turaga, D., & Chen, T. (1999). Correlation based search algorithm for motion estimation. Proc. Picture Coding Symp., Portland, pp. 99-102.
5. Bolton, M., Homewood, F., Robinson, A., Bagni, D., & Borneo, A. (2002). A VLIW processor-based audio/video codec for consumer applications. Proc. Digest of Technical Papers, Int. Conf. on Consumer Electronics, pp. 268-269.
6. Cheung, C.-H., & Po, L.-M. (2002). A novel cross-diamond search algorithm for fast block motion estimation. IEEE Transactions on Circuits and Systems for Video Technology, v. 12, pp. 1168-1177.
7. Cheung, C.-H., & Po, L.-M. (2005). Novel cross-diamond-hexagonal search

algorithms for fast block motion estimation. IEEE Transactions on Multimedia, v. 7, pp. 16-22.

8. Cradle (2004). The Multiprocessor DSP (MDSP) Architecture. Mountain View, CA: Cradle Technologies.
9. Dutta, S., & Wolf, W. (1996). A flexible parallel architecture adapted to block-matching motion-estimation algorithms. IEEE Transactions on Circuits and Systems for Video Technology, v. 6, pp. 74-86.
10. Fanucci, L., Saponara, S., & Bertini, L. (2001). A parametric VLSI architecture for video motion estimation. Integration: The VLSI Journal, v. 31, pp. 79-100.
11. He, Y., Ahmad, I., & Liou, M. L. (1999). Modeling and scheduling for MPEG-4 based video encoder using a cluster of workstations. Proc. 4th Int. Conf. of the Austrian Center for Parallel Computation, Salzburg, Austria, pp. 306-316.
12. ISO/IEC (1995). Information Technology-Generic Coding of Moving Pictures and Associated Audio Information: Video. 13818-2-ITU-T Rec. H.262 (MPEG-2 Video).
13. ISO/IEC (1999). Information Technology-Generic Coding of Audio-Visual Objects: Part 2: Visual. 14496-2 (MPEG-4 Video).
14. ITU-T (1993). Video Codec for Audiovisual Services at p × 64 kbit/s. Rec. H.261.
15. ITU-T (1996). Video Coding for Low Bitrate Communication. Rec. H.263.
16. Kang, J.-Y., Gupta, S., Shah, S., & Gaudiot, J.-L. (2003). An efficient PIM (processor-in-memory) architecture for motion estimation. Proc. 14th IEEE Int. Conf. on Application-Specific Systems, Architecture, and Processors, The Hague, The Netherlands, pp. 282-292.
17. Kittitornkun, S., & Hu, Y.-H. (2001). Frame-level pipelined motion estimation array processor. IEEE Transactions on Circuits and Systems for Video Technology, v. 11, pp. 248-251.
18. Koga, T., Linuma, K., Hirano, A., Iijima, Y., & Ishigr, T. (1981). Motion compensated interframe image coding for video conference. Proc. NTC81, New Orleans, LA, pp. G5.3.1-5.3.5.
19. Kuhn, P. (1999). Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation. Boston, MA: Kluwer.
20. Li, R., Zeng, B., & Liou, M. L. (1994). A new three-step search algorithm for block motion estimation. IEEE Transactions on Circuits and Systems for Video Technology, v. 4, pp. 438-442.
21. Nam, J.-Y., Seo, J.-S., Kwak, J.-S., Lee, M.-H., & Ha, Y. H. (2000). New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector. IEEE Transactions on Consumer Electronics, v. 46, pp. 934-942.
22. Parhi, K. K., & Nishitani, T. (1999). Digital Signal Processing for Multimedia Systems. New York, NY: Marcel Dekker.
23. Po, L. M., & Ma, W. C. (1996). A novel four-step search algorithm for fast block motion estimation. IEEE Transactions on Circuits and Systems for Video Technology, v. 6, pp. 313-317.

24. Shen, K., & Delp, E. J. (1996). A spatial-temporal parallel approach for real-time MPEG video compression. Proc. 1996 Int. Conf. on Parallel Processing, Bloomingdale, IL, pp. II-100-107.
25. Shi, Y. Q., & Sun, H. (1999). Image and Video Compression for Multimedia Engineering. New York, NY: CRC.
26. Tan, M., Siegel, J. M., & Siegel, H. J. (1999). Parallel implementations of block-based motion vector estimation for video compression on four parallel processing systems. International Journal of Parallel Programming, v. 27, pp. 195-225.
27. Tourapis, A. M., Au, O. C., & Liou, M. L. (2000). Fast block-matching motion estimation using predictive motion vector field adaptive search technique (PMVFAST). Proc. ISO/IEC JTC1/SC29/WG11 MPEG2000/m5866, Noordwijkerhout, the Netherlands, pp.
28. Tuan, J.-C., Chang, T.-S., & Jen, C.-W. (2002). On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture. IEEE Transactions on Circuits and Systems for Video Technology, v. 12, pp. 61-72.
29. Vos, L. D., & Schobinger, M. (1993). Efficient architecture of a programmable block matching processor. Proc. Int. Conf. Application-Specific Array Processors, Venice, ITALY, pp. 560-571.
30. Wyland, D. C. (1999). The Universal Micro System: Hardware Performance with Software Convenience. Mountain View, CA: Cradle Technologies.
31. Xu, J.-B., Po, L.-M., & Cheung, C.-K. (1997). A new prediction model search algorithm for fast block motion estimation. Proc. IEEE Int. Conf. Image Processing, Santa Barbara, CA, pp. III-610-613.
32. Zhu, C., Lin, X., & Chau, L.-P. (2002). Hexagon-based search pattern for fast block motion estimation. IEEE Transactions on Circuits and Systems for Video Technology, v. 12, pp. 349-355.
33. Zhu, S., & Ma, K.-K. (2000). A new diamond search algorithm for fast block-matching motion estimation. IEEE Transactions on Image Processing, v. 9, pp. 287-290.