A Comparison of Block-Matching Algorithms Mapped to Systolic-Array Implementation

Sheu-Chih Cheng and Hsueh-Ming Hang, *Senior Member, IEEE*

***Abstract*— This paper presents an evaluation of several well-known block-matching motion estimation algorithms from a system-level very large scale integration (VLSI) design viewpoint. Because a straightforward block-matching algorithm (BMA) demands a very large amount of computing power, many fast algorithms have been developed. However, these fast algorithms are often designed to merely reduce arithmetic operations without considering their overall performance in VLSI implementation. In this paper, three criteria are used to compare various block-matching algorithms: 1) silicon area, 2) input/output requirement, and 3) image quality. A basic systolic array architecture is chosen to implement all the selected algorithms. The purpose of this study is to compare these representative BMA's using the aforementioned criteria. The advantages/disadvantages of these algorithms in terms of their hardware tradeoff are discussed. The methodology and results presented here provide useful guidelines to system designers in selecting a BMA for VLSI implementation.**

***Index Terms*— Architecture mapping, block matching, motion estimation, MPEG-2, systolic array.**

I. Introduction

In designing a very large scale integration (VLSI) chip, there are tradeoffs among various chip cost and performance factors particularly from the system design viewpoint [1]. Since the chip design and layout process is time-consuming and expensive, it is very desirable to be able to predict the overall system performance of a high-level algorithm before its circuit layout is fully deployed. The focus of this paper is to discuss the impact of different block-matching motion estimation (ME) algorithms on VLSI design. Because of the complexity of the entire motion estimation system, decision in choosing one algorithm versus the other algorithms is often empirical and heuristic. For example, the previous motion estimator design often pays attention to only the processor complexity; however, the I/O bandwidth and the on-chip memory size are as important in determining the manufacturing cost.

Motion estimation is an essential element in a standard video coder such as H.261, MPEG1, and MPEG2. A straightforward implementation of a block-matching motion estimation algorithm requires a large amount of hardware. Many *fast* block-matching algorithms have thus been devised to reduce the computational complexity without degrading the estimation performance significantly. Examples of fast algorithms are described in [2]–[6]. We choose six well-known algorithms and analyze them in depth in this paper. They are the exhaustive search, the three-step search, the modified log search, the conjugate direction search, the alternating pixel-decimation search, and the subsampled motion-field search. Although these algorithms are devised to use fewer arithmetic operations, they may need additional control circuits and data buffers and thus may not lead to lower cost in VLSI manufacturing.

The aforementioned algorithms are chosen not only because of their popularity, but also because they are rather generic and they represent different ways of cutting down the computation. The decimation search reduces the number of data points in each matching calculation, while the other searches try to reduce the number of search points using different techniques (explained in Section II). There are many variations of these algorithms. For example, we could compare the partial results against a properly selected threshold and terminate the searching process in the middle to save computation [22]. In addition, there exist many other block-matching algorithms (BMA's) that we cannot cover here. One structure worth mentioning is the hierarchical algorithm that performs a search first on a loose grid and then on a denser grid [4], [5]. The hierarchical steps can be more than two. In a way, it is a variation of a sequential search (like three-step search). Nevertheless, the analysis described in this paper can be applied to the algorithms not included here.

The hardware implementation of motion estimation algorithms can be classified into programmable video signal processor (VSP) structures and dedicated (special purpose) structures. Programmable VSP structures [7]–[9] allow a higher degree of flexibility; however, they often have a lower throughput rate, higher hardware cost [10], and generally require additional software development effort. Using today's fabrication technology, dedicated structures seem to be more economical for mass production. Therefore, we consider only the dedicated structure in this paper.

Typically, a specific motion estimation algorithm is first chosen and then a specific hardware architecture is designed for this chosen algorithm. For example, several hardware implementations are designed for the exhaustive search algorithm [11]–[13] and a couple of implementations for the fast algorithms [14], [15]. Also, a few programmable architectures [16]–[18] have been proposed and designed to implement both the exhaustive and some selected fast search algorithms. Usually, these architectures require additional special control

Manuscript received September 30, 1996; revised January 31, 1997. This work was supported in part by the National Science Council of R.O.C. under Grant NSC86-2221-E-009-023.

The authors are with the Department of Electronics Engineering and Center for Telecommunications Research, National Chiao Tung University, Hsinchu, Taiwan 300, R.O.C.

Publisher Item Identifier S 1051-8215(97)05884-9.

circuit and memory management to set up data paths for different algorithms.

For a specific algorithm such as the sequential (or hierarchical) algorithm, a well-designed programmable architecture could be rather efficient [18]. However, identifying the *optimal* VLSI design for every BMA of interest is a very difficult task. One possible approach is collecting all the existing architectures and comparing them. For example, Pirsch *et al.* [10] has an excellent summary of the existing ME chips (but their analysis is focused on the comparison of different *implementations*). Even if following this approach, we are not sure we could claim our choice for Algorithm A is as *optimal* as that for Algorithm B. Hence, a different approach is taken. We choose a hardware structure as the common ground for comparing all algorithms. Because of its regular layout, high throughput rate, and massive parallel computing capability, systolic array is a very popular and good candidate structure in designing motion estimation chips [11], [19]. Therefore, we also use the systolic array architecture as the basic building block in implementing various block-matching algorithms.

Our goal in this paper is evaluating block matching algorithms from mainly the hardware viewpoint. We try to point out that the traditional measure of algorithms, the number of operations, does not match well the VLSI performance. Also, the known fast algorithms have significant VLSI advantages only for large search ranges and large size pictures. We do not intend to find the *best* algorithm and architecture combination in this study. We compare only algorithms using essentially the same basic hardware structure. Some of the above observations were touched in the past, but we have not seen reports with thorough studies.

The information contained in this paper may serve as a reference or guide to system designers. Given a specific application (picture size, search range, etc.) a hardware designer can start with a couple of the more promising algorithms and tune the architecture/hardware layout for that specific algorithm. An algorithm (and architecture) designer can also learn from this study what elements are more critical in a BMA for improving VLSI performance and thus designs algorithms accordingly. Furthermore, this work is an attempt to propose a methodology in evaluating algorithms from both VLSI implementation viewpoint and compression performance. A similar study can be applied to the other block matching algorithms and other types of signal processing algorithms. On the other hand, our approach is limited by the varying efficiency of the proposed structure on different algorithms. However, our survey on the existing BMA VLSI structures indicates that this set of implementations should be able to show the distinct advantages and disadvantages of various algorithms in VLSI implementation.

The rest of this paper is organized as follows. Section II describes the block-matching algorithms examined in this paper. Section III discusses the systolic array structures for the evaluated algorithms and their computational complexity. In Section IV, we look into the silicon cost and I/O configuration issues for different algorithms in various applications. Section V shows the simulation results of picture quality of the examined algorithms. Section VI briefly summarizes our work in this paper.

TABLE I
MOTION ESTIMATION PARAMETERS FOR CCIR-601 AND CIF PICTURES

Parameters	Symbol	CCIR-601	CIF
Picture size	$P_h \times P_v$	720×480	352×288
Picture rate	f_r	30	10
Block size	$N \times N$	16×16	16×16
Maximum search range	ω	47	7 or 15
Number of blocks per second	K	40500	3960

II. BLOCK-MATCHING MOTION ESTIMATION ALGORITHMS

Block-matching motion estimation is an effective method in reducing the temporal redundancy in video coding and thus is adopted by many video coding standards [2], [5]. The basic operation of a block-matching algorithm is picking up the best candidate image block in the reference image frame by calculating and comparing the matching functions between the current image block and all the candidate blocks inside a confined area in the reference frame. The sizes of image block and confined area (so-called *search area*) have a strong impact on the performance and the computational complexity of the motion estimation results. A small size block offers a good approximation to the moving object, but it also produces a large amount of redundant motion information data. Small size blocks are easily interfered by random noise. On the other hand, large size blocks may produce a less accurate motion vector since a large block may contain two or more objects moving at different speeds and directions. Block sizes of 8×8 or 16×16 are generally considered adequate from experiments, and thus the international video standards adopt the 16×16 block size, which is used in this study.

To decide an adequate search area is somewhat involved. It depends on both the contents of pictures and the coding system structure. For video-phone applications, small pictures and slow motion are expected, and thus the search range is assumed to be small (around 7 or 15 pels). On the other hand, in MPEG coding, large pictures are expected and the temporal distance between two predictive frames (P-frames) is often greater than a couple of frames [20]. Hence, a large search range (say, 47 pels) is necessary. In addition to block size and search range, picture size and frame rate also have a strong impact on the VLSI cost.

In summary, the important parameters used in the following discussions are: i) picture size (horizontal and vertical), $P_h \cdot P_v$; ii) picture rate (frames/s), f_r; iii) block size, $N \cdot N$; iv) search range, ω; v) external memory bus width, W; and vi) the number of image blocks per second which is derived from the first three parameters: $K = (P_h/N) \cdot (P_v/N) \cdot f_r$. The parameters used in this paper are listed in Table I for CCIR-601 and common intermediate format (CIF) pictures. The former picture format is targeting at digital television (DTV) applications and the latter, video-phone applications.

Another important factor that affects the block-matching hardware complexity is the matching criterion. To reduce computational complexity, the mean absolute difference (MAD) criterion is adopted by almost all the VLSI designs in the market and in the literature. It (MAD) provides a motion estimation performance nearly comparable to the more complicated matching criteria such as the mean square error [2], [5]. Some fast search algorithms calculate the frame differences only on the decimated pels (described in Sections II-E and II-F). For convenience, we thus define two terms: *SAD (sum of absolute difference)* is referred to the ordinary MAD performed on every pel inside a block, and *SDAD (sum of decimated absolute difference)* is the MAD that applies to only the decimated pels. That is, for $-\omega \leq u, v \leq \omega - 1$

$$\mathrm{SAD}(u, v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathrm{AD}(x, y, u, v) \tag{1}$$

$$\mathrm{SDAD}(u, v) = \sum_{(x, y) \in \text{decimation pattern}} \mathrm{AD}(x, y, u, v) \tag{2}$$

$$\mathrm{AD}(x, y, u, v) \equiv |f(x, y, t) - f(x-u, y-v, t-\Delta t)|$$

where $f(x, y, t)$ and $f(x-u, y-v, t-\Delta t)$ are the pel values in the current block and in the reference (frame) block, respectively, (x, y) is the pel coordinate relative to the current block location, (u, v) is the (backward) motion vector, and Δt is the time difference (temporal distance) between the current and the reference frames.

The final motion vector is the one that minimizes the MAD criterion

$$\min_{u, v} \{\mathrm{SAD}(u, v) \text{ or } \mathrm{SDAD}(u, v)\} \quad -\omega \leq u, v \leq \omega - 1. \tag{3}$$

In general, we need N^2 subtractions, N^2 absolute operations, and $N \cdot (N-1)$ additions to compute one point of SAD (u, v). For SDAD, all the aforementioned operations are reduced by a factor determined by the chosen decimation pattern. In the alternate pixel decimation algorithm described in this paper, the reducing factor is four. For the N values in the range of interest such as $N = 16$, $N(N-1)$ can be reasonably approximated by N^2. The total number of operations needed to compute the MAD criterion is an important attribute of an ME algorithm. In the rest of this section, we briefly describe the operations of the motion estimation algorithms examined in this paper.

A. Exhaustive Search

The most straightforward searching algorithm is the exhaustive search (full search), which evaluates all the possible displacements (motion vector candidates) inside the search area. In each block time interval, $(2\omega+1)^2$ SAD search points and $(2\omega+1)^2$ two-term comparisons are calculated to find the best match. In other words, its computational complexity is on the order of ω^2, $O(\omega^2)$.

B. Three-Step Search

This popular fast search algorithm is proposed by Koga *et al.* [21]. It starts with a step size slightly larger than half of the search range. In the first step, the algorithm compares and selects the minimum SAD from the nine candidate locations located on the corners and the midpoints of the square borders one step size away from the center. The minimum point becomes the center of the next step. In the second step, the step size is halved and eight new candidates located again on the square borders are calculated. The new minimum point is obtained by comparing the SAD values of the new eight candidates together with the previous minimum. The above procedure is repeated until the step size is smaller than one and the final motion vector is thus found. Unfortunately, unlike the exhaustive search algorithm, the candidate points are data-dependent—the current step result decides the to-be-evaluated search points in the next step. Therefore, each step has to be performed sequentially. In total, there are $\log_2(\omega+1)$ search steps and $1+8\log_2(\omega+1)$ SAD search points for each image block. It is clear that the number of search steps and points must be an integer and thus $\log_2(y)$ denotes the least integer greater than or equal to $\log_2(y)$ in the rest of this paper.

C. Modified Log Search

This fast search algorithm is proposed by Kappagantula and Rao [22]. The procedure in this algorithm is similar to that of the three-step search but each search step is broken into two substeps. In the first substep, five search points are evaluated. They consist of the central point of a diamond-shape region and the four search points located one step size away from the central point along the horizontal and vertical directions. If the minimum-SAD position is the central point, the step size is halved and the above process is repeated again. Otherwise, one of the corner points is the minimum point and the second substep is activated. Two additional search points located one step size away from the minimum point are evaluated. These two new search points are located vertically if the first substep minimum point is on the horizontal line. Otherwise, two horizontal search points are used. The minimum among these three search points becomes the center of the new diamond-shape region with a step size equal to half of the previous step size. Then, the next search step starts. The above procedure continues until the step size is smaller than one. The number of SAD calculations in this algorithm varies depending upon the location of the final motion vector. However, we need to consider the worst case situation in VLSI design, and thus there are $1+6\log_2(\omega+1)$ SAD operations for each block.

D. Conjugate Direction Search

The conjugate direction search algorithm suggested by Srinivasan and Rao [23] breaks the two-dimensional (2-D) search problem into two one-dimensional (1-D) problems. Assuming that the search starts with the horizontal direction, we first compute the SAD of three candidates located one next to the other. The center candidate is typically the zero motion vector. Then, compare and select the minimum SAD from these three values. If the minimum-SAD position is not the central point, it becomes the new center and the position immediately next to it along the minimum-SAD direction is included as the new candidate. The above procedure is repeated until

a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d
a	b	a	b	a	b	a	b
c	d	c	d	c	d	c	d

Fig. 1. Decimated patterns for computing SDAD.

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

Fig. 2. Alternating patterns of pels in the search region for the APD technique.

the minimum-SAD position is the central point or we hit the search area boundary. In either case, the horizontal direction is completed and we turn to the vertical direction. Starting from the current minimum-SAD point, the same procedure is applied to find the vertical minimum point. The number of operations in this algorithm depends on the location of the final motion vector. In the worst case, there are $2\omega + 3$ SAD operations for each block. When the search range (ω) is very large, its computational complexity is on the order of ω, $O(\omega)$, larger than that of the three-step search whose computational complexity is $O(\log_2 \omega)$.

E. Alternating Pixel-Decimation (APD) Search

This algorithm is proposed by Liu and Zaccarin [24]. It differs from the previous fast algorithms in that it tries to reduce the calculations involved in each SAD operation but maintains the overall motion estimation performance at a comparable level. The basic concept is to decimate the pels inside a block and compute the differences only on the decimated pels. This algorithm can be explained by using Figs. 1 and 2. Fig. 1 shows a block of 8 × 8 pels with pels labeled **a**, **b**, **c**, and **d** in a regular manner. The decimation pattern **A** is made of all the **a** pels. Patterns **B**, **C**, and **D** are similarly defined. Fig. 2 shows the pels in (a portion of) the search area. They are labeled **1**, **2**, **3**, and **4**. For example, when a **1** pel is a motion vector candidate, pattern **A** is used as the decimation pattern to pick up the pels in calculating SDAD. Similarly, patterns **B**, **C**, and **D** are the decimation patterns for the candidates located at pels **2**, **3**, and **4**, respectively. For each of these four decimation patterns, the minimum SDAD candidate is retained. Then, for each decimation pattern, the full SAD is computed using all the block pels. The best among them becomes the final motion vector.

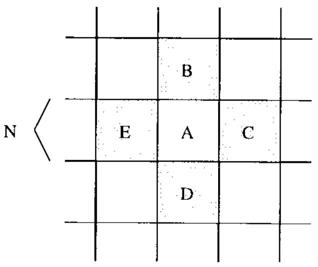

Fig. 3. White and shaded blocks in the SAPD technique.

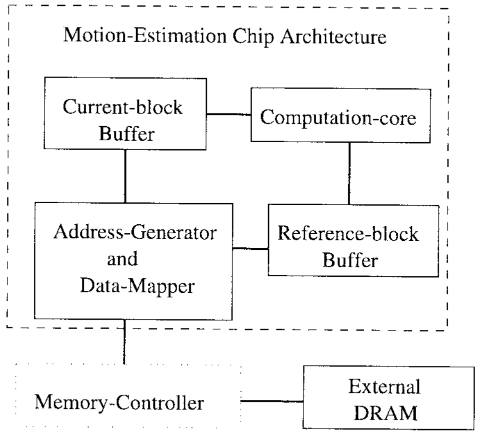

Fig. 4. Block diagram of a general motion estimation chip.

In the above procedure, $(2\omega + 1)^2$ SDAD operations are needed for one block. In addition, four SAD operations are calculated for the final motion vector selection. If the four SAD operations are neglected for large search ranges, the computational complexity of this algorithm is roughly a quarter of that of the exhaustive search. Hence, its computational complexity is $O(\omega^2)$, the same as the exhaustive search.

F. Subsampled Motion-Field Search with Alternating Pixel-Decimation Patterns (SAPD)

This algorithm combines both motion field subsampling and the alternating pixel decimation (APD) techniques [24]. There are two stages. At the first stage, we estimate half of the motion vectors using the previous APD technique. The locations of the estimated blocks are indicated by the shaded blocks in Fig. 3. At the second stage, the motion vector of a white block is calculated based on the four vectors of its adjacent shaded blocks. For example, the motion vector assigned to the white block **A** in Fig. 3 is one of the motion vectors of blocks **B**, **C**, **D**, or **E** that gives the smallest SAD value.

In the original formulation [24], block **A** could be used as a *subblock* to increase the motion estimation accuracy. However, in order to match the MPEG coding structure, block **A** in this paper has the size of the basic motion estimation unit, $N \cdot N$. Thus, for a shaded block, $(2\omega + 1)^2/2$ SDAD and four SAD operations are needed to compute its motion vector. In addition, four SAD operations are needed for each white block. The total computational complexity of this algorithm is approximately reduced by a factor of eight in comparing with

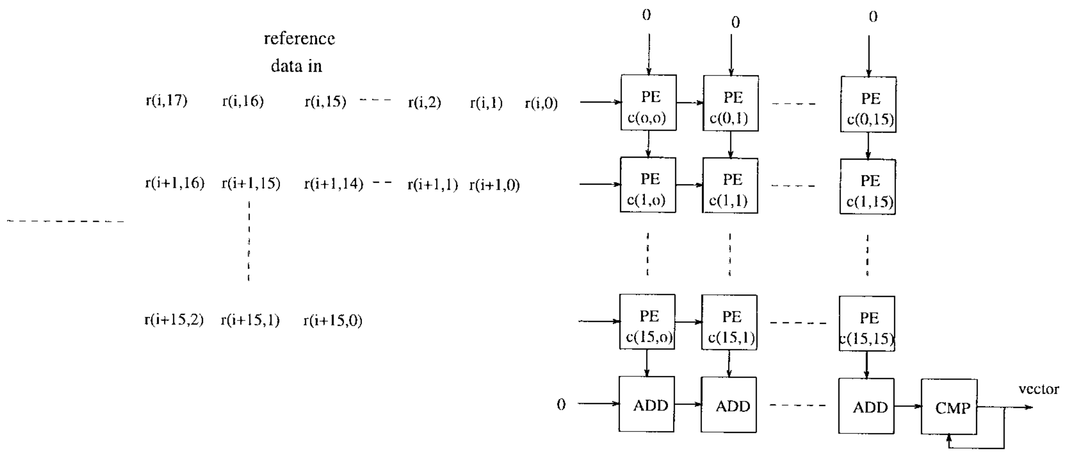
reference data in
r(i,17) r(i,16) r(i,15) r(i,2) r(i,1) r(i,0)
r(i+1,16) r(i+1,15) r(i+1,14) r(i+1,1) r(i+1,0)
r(i+15,2) r(i+15,1) r(i+15,0)
0
PE c(0,0)
PE c(0,1)
PE c(0,15)
PE c(1,0)
PE c(1,1)
PE c(1,15)
PE c(15,0)
PE c(15,1)
PE c(15,15)
ADD
CMP
vector

Fig. 5. Block diagram of the 2-D systolic architecture for block-matching.

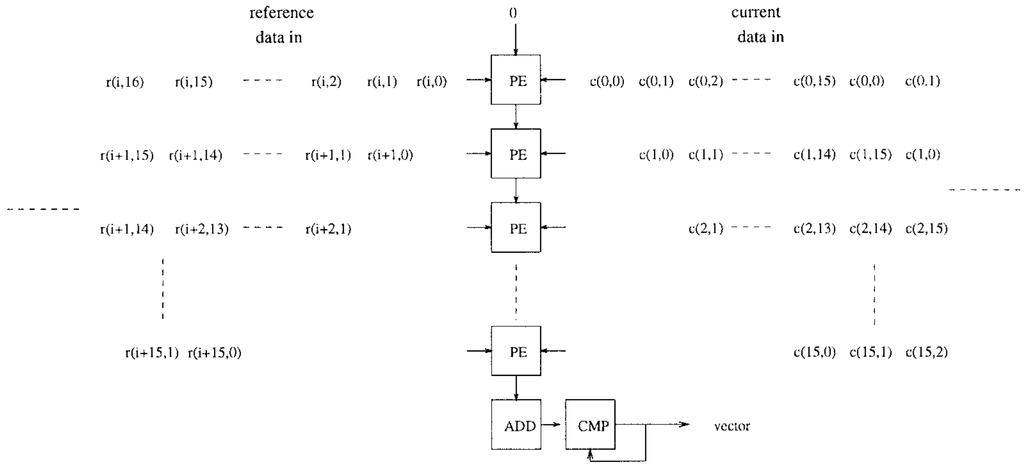
reference data in
current data in
r(i,16) r(i,15) r(i,2) r(i,1) r(i,0)
r(i+1,15) r(i+1,14) r(i+1,1) r(i+1,0)
r(i+1,14) r(i+2,13) r(i+2,1)
r(i+15,1) r(i+15,0)
c(0,0) c(0,1) c(0,2) c(0,15) c(0,0) c(0,1)
c(1,0) c(1,1) c(1,14) c(1,15) c(1,0)
c(2,1) c(2,13) c(2,14) c(2,15)
c(15,0) c(15,1) c(15,2)
0
PE
ADD
CMP
vector

Fig. 6. Block diagram of the one-column systolic architecture.

the exhaustive search. But its computational complexity is still $O(\omega^2)$, the same as the exhaustive search.

III. VLSI Implementation and Complexity Analysis

Several important factors have to be considered in choosing an algorithm for VLSI implementation, for example, i) chip area, ii) I/O bandwidth, and iii) image quality. We will discuss the first two factors in this and the next sections and the third factor in Section V. In implementing block-matching algorithms, the chip area can be approximated by

$$A_{\text{total}} = A_{cp} + A_{buf} + A_{ctl} \tag{4}$$

where A_{cp} is the area used for the computation kernel, A_{buf} is for the on-chip data buffer, and A_{ctl} is for the system controller. Because of the massive local connection and parallel data flow in the systolic array structure, a system controller is needed to generate data addresses and flow control signals. Particularly, computing SAD requires specific ordering of data. Therefore, our system controller contains an address generator and a data flow controller.

Due to the very massive data used in computing motion vectors, it becomes impractical for the processor array to access image data directly from the external memory for it results in a very high bus bandwidth requirement. In addition, the search areas of nearby blocks overlap significantly; hence, an internal reference-data buffer is introduced to relieve some of the external memory access. The block diagram of our motion estimation system with internal buffer is shown in Fig. 4. The memory controller reads in the current and the reference image blocks from external DRAM and stores them in the *current-block buffer* and the *reference-frame buffer*, respectively. In this paper, the I/O configuration is referred to as the number of I/O pads and the I/O speed requirement which is constrained by the external memory speed. The external memory bandwidth depends on the size of the internal buffer. This topic will be elaborated in Section IV.

A. Mapping Algorithms to Architectures

Systolic architectures are good candidates for VLSI realization of block-matching algorithms with a regular search procedure [19]. A typical systolic array consists of local connections only and thus does not require significant control circuitry overhead. In this paper, a basic systolic array architecture is adopted for estimating the silicon area of various block-matching algorithms. Its general structure is shown in

Fig. 5. The processor array (2-D array architecture) consists of 16 $\times$ 16 processor elements (PE's) or 8 $\times$ 8 PE's if the APD search technique is in use. If the number of PE's (N_{PE}) is less than or equal to 16, then this system is reduced to one-column architecture as shown in Fig. 6. If the estimated N_{PE} is smaller than the size of a 2-D array but larger than that of a 1-D array (one-column), we then use multiple one-column circuits. In the multiple one-column structures, independent data are processed by several one-column circuits simultaneously. Similarly, multiple 2-D arrays are used when N_{PE} is several times larger than the size of a 2-D array. Four types of computing nodes are used in this structure. Their circuits are shown in Fig. 7. The subtraction, absolute value, and partial sum addition in SAD or SDAD are performed by the PE node. The summation operations are done by the ADD nodes. The CMP nodes compare the matching criteria of the candidates and select the minimum one. The AP node is used to execute the operations of both ADD and CMP when the speed requirement is not critical.

In the 2-D array structure, the current block data, $c(k, l), k, l = 0, \cdots, 15$, are first loaded into each PE node. Then, the reference block data, $r(i + k, l)$, slide in from the left. The calculation starts from the upper-left corner of the processor array. During the first clock cycle, the $c(0, 0)$ node computes the absolute difference between $r(i, 0)$ and $c(0, 0)$. The result passes to the PE node below. During the second clock cycle, the $c(1, 0)$ node computes the absolute difference between $r(i + 1, 0)$ and $c(1, 0)$ and adds its result to the partial sum propagated from above. In the meanwhile, node $c(0, 0)$ computes the absolute difference between $r(i, 1)$ and $c(0, 0)$, and node $c(0, 1)$ computes the absolute difference between $r(i, 0)$ and $c(0, 1)$. After 16 clock cycles, the first partial sum, $\sum_{k=0}^{15} |r(i + k, 0) - c(k, 0)|$ is completed and placed into the left-most ADD node. In the following clock cycle, this partial sum is passed to the immediate right ADD node and added together with the second partial sum, $\sum_{k=0}^{15} |r(i + k, 1) - c(k, 1)|$. The total sum (SAD) for the motion vector candidate $(i, 0)$ is completed and propagated to CMP in the following 14 clock cycles. This SAD is compared against the stored SAD resulting from the previous comparison, and then the smaller one is kept in CMP for future comparison. The preceding computation procedure is repeated until all possible candidates are compared and the final motion vector is obtained. The one-column array computation procedure is similar. Some algorithm variations can be implemented with a small addition to the CMP node. For example, the CMP node can be modified to a two-stage structure that compares the calculated SAD with a preselected threshold value at the first stage and then performs the ordinary comparison against the previous matched value. The search process terminates if the calculated SAD is smaller than the threshold. Thus, we realize the "stopping in the middle" feature.

It is clear that an address generator is needed to generate the proper addresses to retrieve data, and then these data have to be distributed properly by a data flow controller (DFC) to the processor array at correct timing. Fig. 8 shows the block diagram of DFC. The output data are broadcasted to the assigned PE nodes during the current clock cycle and then propagated to successive PE nodes in the next clock. The efficiency (EFF) of an array architecture is defined to be the ratio of the active operating time (of all PE's) to the total operating time (including the idling time for data loading).

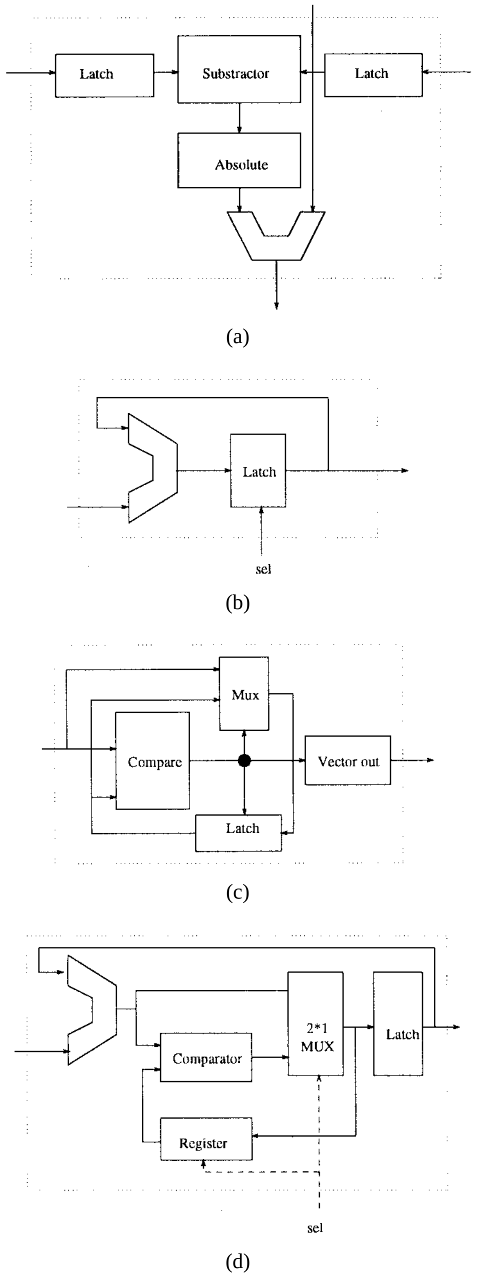

Fig. 7. Schematic diagrams of PE, ADD, CMP, and AP node elements. (a) block diagram of "PE" node, (b) block diagram of "ADD" node, (c) block diagram of "CMP" node, and (d) block diagram of "AP" node.

The silicon area of the computation kernel used in this architecture can be approximated by

$$A_{cp} = N_{\mathrm{PE}} \cdot A_{\mathrm{PE}} + N_{\mathrm{ADD}} \cdot A_{\mathrm{ADD}} + N_{\mathrm{CMP}} \cdot A_{\mathrm{CMP}} \quad (5)$$

where N_{PE}, N_{ADD}, and N_{CMP} are the numbers of PE, ADD, and CMP nodes, respectively; A_{PE} is the silicon area of one PE, and A_{ADD} and A_{CMP} are similarly defined. In this architecture, the number of PE's is decided by clock rate, picture size, and search range. If one-column array is sufficient

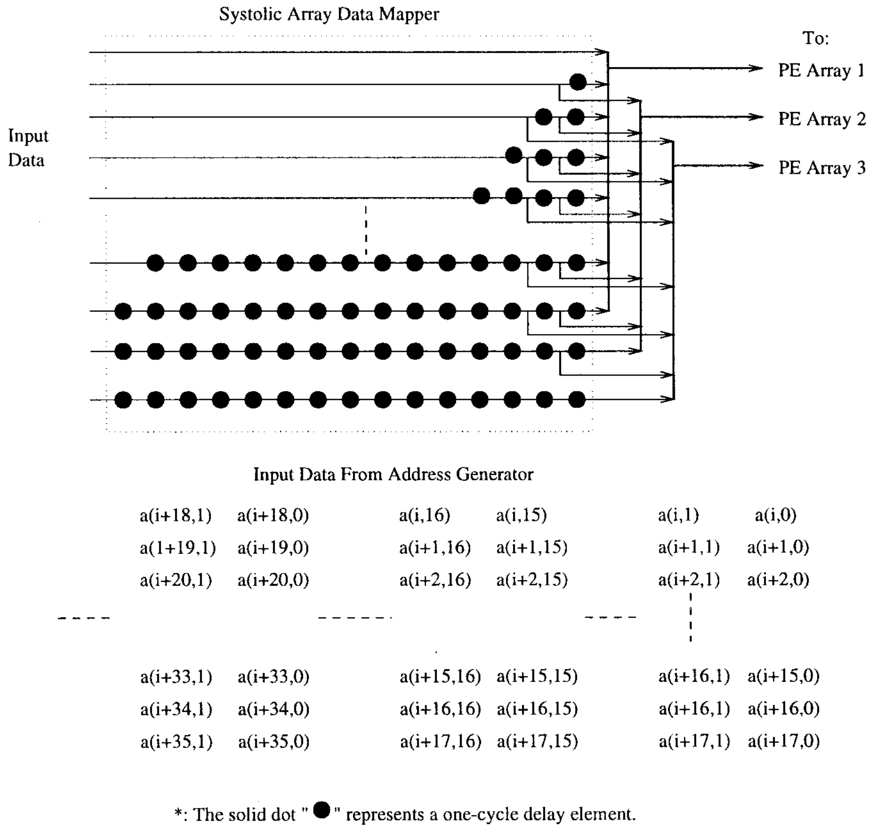
Systolic Array Data Mapper
To:
PE Array 1
PE Array 2
PE Array 3
Input Data
Input Data From Address Generator
a(i+18,1) a(i+18,0) a(i,16) a(i,15) a(i,1) a(i,0)
a(1+19,1) a(i+19,0) a(i+1,16) a(i+1,15) a(i+1,1) a(i+1,0)
a(i+20,1) a(i+20,0) a(i+2,16) a(i+2,15) a(i+2,1) a(i+2,0)
a(i+33,1) a(i+33,0) a(i+15,16) a(i+15,15) a(i+16,1) a(i+15,0)
a(i+34,1) a(i+34,0) a(i+16,16) a(i+16,15) a(i+16,1) a(i+16,0)
a(i+35,1) a(i+35,0) a(i+17,16) a(i+17,15) a(i+17,1) a(i+17,0)
*: The solid dot " ● " represents a one-cycle delay element.

Fig. 8. Block diagram of data flow controller.

to process the data in time, it will be chosen to increase the utilization efficiency (EFF) of PE. Otherwise, the 2-D array is forced into use. The PE number, N_{PE}, is also restricted by the maximum system clock. To match the available IC fabrication technology, the maximum clock rate is assumed to be 100 MHz for DTV and 40 MHz for video-phone applications.

Search areas of adjacent blocks overlap quite significantly. This overlapped area data can be stored inside the internal (on-chip) buffer to reduce external memory accesses (bandwidth). Three types of internal buffers for the exhaustive and the APD searches are under evaluation: i) Type A buffer whose size equals to the search area, $(N+2\omega)\cdot(N+2\omega)$ pels; ii) Type B buffer that has the size of one slice of search area; that is, the height of block (or subblock) times the width of search area, $[(N+2\omega)/D]\cdot[N/D]$ pels; and iii) Type C buffer that has the size of a block or a subblock, $(N/D)\cdot(N/D)$ pels. Note that the parameter D in the above expressions equals one for SAD calculation and equals two for SDAD (pel decimation technique). For the other search schemes, Type A and C buffers are still meaningful. However, the Type B buffer defined here does not always make sense for sequential searches. Therefore, we may modify the size and the function of the Type B buffer when appropriate. Generally, we assume that the Type B buffer can hold the data needed for processing one search step. This, in fact, in certain cases does not save either computation or bus bandwidth as will be noted.

A picture frame contains P_v/N picture slices and each slice contains P_h/N blocks. In order to derive the I/O bandwidth requirement, we first calculate the size of the new data to be loaded from the external memory down to the on-chip buffer for each block. As shown in Fig. 9(a), the newly loaded data size for the Type A buffer is $N\cdot(N+2\omega)$ pels when the next block is on the same picture slice. For processing one picture slice, we need to load the complete buffer at the beginning of a slice; thus, the total external data access is approximately $(N+2\omega)^2+(P_h/N-1)\cdot N\cdot(N+2\omega)$ pels if boundary block cases are neglected. Then, for the entire picture, the total external data access is approximately $f_r\cdot(P_v/N)\cdot[(N+2\omega)^2+(P_h/N-1)\cdot N\cdot(N+2\omega)]$ pels. Similar analysis can be carried over to the cases of Type B and C buffers as shown in Fig. 9(b) and (c). The exact sizes of Type B and C buffers depend on the search algorithms and will be discussed in the next subsection. Either one-port or two-port on-chip memory can be used as the internal buffer. The two-port buffer has the advantage of having higher processor utilization efficiency because data read and write can be executed in the same clock cycle. The drawback is it costs more silicon area to implement.

B. Computational Complexity

In this section, we discuss the computational complexity of the motion estimation algorithms described in Section II. There are two stages in loading the reference block data from the off-chip RAM to the systolic array. In the first or the *external* stage, data are moved from the off-chip RAM to the on-chip buffer, and in the second or the *internal* stage, data are moved from the on-chip buffer to the systolic array

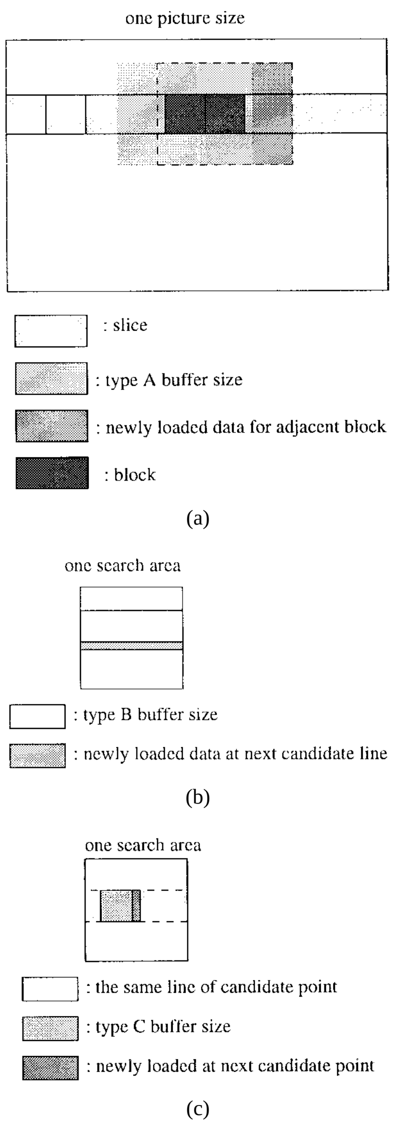
one picture size
: slice
: type A buffer size
: newly loaded data for adjacent block
: block
(a)
one search area
: type B buffer size
: newly loaded data at next candidate line
(b)
one search area
: the same line of candidate point
: type C buffer size
: newly loaded at next candidate point
(c)

Fig. 9. The block diagrams of overlapped area for three types of buffers. (a) Type A buffer, (b) Type B buffer size, and (c) Type C buffer.

processors. There are two issues involved here. The first issue is the external bus bandwidth requirement so that the external data can meet the demand of the processors, and the second issue is the idling time (stuffing cycles) that the processors are waiting for the data to arrive. Suppose that the first issue has been solved; that is, the external bus bandwidth is wide enough to supply data in time. Under this assumption we investigate the data lag issue.

The total delay time between the go-signal and the time that the processors produce the first output is the external stage delay plus the internal stage delay. This delay time may not introduce processor stuffing cycles if the data can be preloaded into the pipeline and no gap is needed in and between search processes. This is true for the exhaustive search since the data (reference blocks) locations are known ahead of time, and thus the next block data can be preloaded during the current block interval. In the sequential searches, however, the data locations to be used at the next search step are unknown until the current step is completed. In other words, we start loading data to the systolic array after the previous search step is completed. There is a gap (stuffing cycles) between two search steps. At the beginning of a new search, it takes N_{PE} cycles to fill up the entire systolic array and then two more cycles are needed for the summation and comparison operations. Therefore, our estimate of the stuffing cycles between the completion of the current search step and the first output of the next search step is $N_{\text{PE}} + 2$. There are specific processing structures that can shorten this gap [17]; however, we do not fine-tune our structure for this purpose because our goal is not to design the *optimal* hardware architecture for any particular algorithm. In reality, a couple of cycles should be added due to the external stage delay, but they are small in number and thus are neglected.

1) Exhaustive Search: The computational kernel of this algorithm needs to perform at least $(2\omega+1)^2$ SAD operations in each block time. If we use only one PE, the clock rate has to be higher than 93.57 GHz for encoding a CCIR-601 4 : 2 : 2 resolution picture with a search range of 47 pels. This is impractical. Typically, the maximum clock speed is upper bounded by the fabricating technology and the I/O limitation. To make our analysis more general, we assume an X-MHz clock being employed. The efficiency of systolic architecture for the exhaustive search is nearly 100% because the input data flow is regular and can be arranged in advance. In this case, the number of total PE operations per second is

$$OP_{\text{FUL}} = N_{\text{SAD}} \cdot N^2 \cdot K = (2\omega+1)^2 \cdot N^2 \cdot K \tag{6}$$

where N_{SAD} is the number of SAD operations for one image block, and K is the number of blocks per second (in Table I). Thus, the number of PE nodes required in this structure under the maximum system clock constraint becomes

$$N_{\text{PE}} = \frac{OP_{\text{FUL}}}{X} = \frac{K \cdot (2\omega+1)^2 \cdot N^2}{X}. \tag{7}$$

Here, we assume that multiple copies of systolic structures can be used. The actual N_{PE} value is rounded up to the nearest multiple of 16^2 (2-D array) or 16 (1-D array). The exact number is determined by the picture size, search range, etc., as to be discussed in the following subsections.

We next consider the on-chip (internal) buffer size and the data input rate. The Type A buffer situation has been discussed in Section III-A. In the case of Type B buffer, it first stores N horizontal lines and it then loads one horizontal line when the search moves vertically down one line. In total, 2ω additional horizontal lines have to be loaded for the entire search area and each line contains $N+2\omega$ pels. Therefore, the total input data for computing one block is about $(N+2\omega)^2$ pels. For the Type C buffer, there are $2\omega+1$ candidate positions on the same line, and in this situation, the new data size for the next position on the same line is N pels. Furthermore, the initial data loading for every line is $N \cdot N$ pels. Thus, finishing one line of candidates requires loading $(N \cdot N + N \cdot 2\omega)$ pels. Because there are $2\omega+1$ lines of candidates in one search area, the total input data size for one block is $(2\omega+1) \cdot [N \cdot (N+2\omega)]$ pels. The input data rate and buffer sizes under different configurations are listed in Table II. They have a strong impact on the silicon size and the I/O bandwidth as will be discussed in Tables IV,

TABLE II
(a) IMPLEMENTATION COMPLEXITY (1)

Algorithm (1)		Exhaustive search		
Buffer type size (bytes)	Input data rate (bytes/sec)	A	$(N+2\omega)^2$	$f_r \cdot \frac{P_v}{N} \cdot [(N+2\omega)^2 + (\frac{P_h}{N} - 1) \cdot N \cdot (N+2\omega)]$
		B	$(N+2\omega) \times N$	$(N+2\omega)^2 \cdot K$
		C	$N \times N$	$(2\omega+1) \cdot [N \cdot (N+2\omega)] \cdot K$
Estimated complexity		sub/abs/add		$N^2 \cdot (2\omega+1)^2 \cdot K$
		compare		$(2\omega+1)^2 \cdot K$
N_{PE}	$\frac{N^2 \cdot (2\omega+1)^2 \cdot K}{X}$			
Algorithm (2)		Three step search		
Buffer type size (bytes)	Input data rate (bytes/sec)	A	$(N+2\omega)^2$	$f_r \cdot \frac{P_v}{N} \cdot [(N+2\omega)^2 + (\frac{P_h}{N} - 1) \cdot N \cdot (N+2\omega)]$
		B_1	$9N^2$	$8N^2 \cdot \log_2(\omega+1) \cdot K;\ for\ \omega \geq 4N-1$
		B_2	$(2\lceil\frac{\omega+1}{2}\rceil + N)^2$	$(\frac{\omega+1}{2} + N)^2 \cdot \log_2(\omega+1) \cdot K$; otherwise
		C	$N \times N$	$8N^2 \cdot \log_2(\omega+1) \cdot K$
Estimated complexity		sub/abs/add		$N^2 \cdot (1 + 8\log_2(\omega+1)) \cdot K$
		compare		$(1 + 8\log_2(\omega+1)) \cdot K$
N_{PE}	$\frac{X' - \sqrt{(X')^2 - 4\log_2(\omega+1) \cdot (1+8\log_2(\omega+1)) N^2 K^2}}{2K\log_2(\omega+1)}$; $X' = X - 2\log_2(\omega+1) \cdot K$			
Algorithm (3)		Modified log search		
Buffer type size (bytes)	Input data rate (bytes/sec)	A	$(N+2\omega)^2$	$f_r \cdot \frac{P_v}{N} \cdot [(N+2\omega)^2 + (\frac{P_h}{N} - 1) \cdot N \cdot (N+2\omega)]$
		B_1	$9N^2$	$8N^2 \cdot \log_2(\omega+1) \cdot K;\ for\ \omega \geq 4N-1$
		B_2	$(2\lceil\frac{\omega+1}{2}\rceil + N)^2$	$(\frac{\omega+1}{2} + N)^2 \cdot \log_2(\omega+1) \cdot K$; otherwise
		C	$N \times N$	$8N^2 \cdot \log_2(\omega+1) \cdot K$
Estimated complexity		sub/abs/add		$N^2 \cdot (1 + 6 \cdot \log_2(\omega+1)) \cdot K$
		compare		$1 + 6 \cdot \log_2(\omega+1) \cdot K$
N_{PE}	$\frac{X' - \sqrt{(X')^2 - 8\log_2(\omega+1) \cdot (1+6\log_2(\omega+1)) N^2 \cdot K^2}}{4K\log_2(\omega+1)}$; $X' = X - 4\log_2(\omega+1) \cdot K$			

(a)

VI, VIII, and IX. The numbers of "add," "sub," "abs," and "compare" operations in this table are estimated based on the SAD operations needed as described in Section II.

2) Three Step Search: We still use the same basic systolic structure described in the previous subsection to implement this algorithm. There are $1 + 8\ \log_2(\omega + 1)$ SAD operations to be performed for each block. Therefore, the total number of required PE operations per second is

$$OP_{\mathrm{TSS}} = K \cdot [1 + 8\ \log_2(\omega+1)] \cdot N^2. \tag{8}$$

If we choose the one-column architecture, $N_{\mathrm{PE}} + 2$ pels have to be loaded into the processor array between two search steps, because we do not know which data is to be processed until the completion of the current step. Thus, the efficiency of the one-column structure is approximately

$$\begin{aligned} &EFF_{\mathrm{TSS}} \\ &= \frac{N_{act\_ck} \cdot N_{\mathrm{SAD}}}{N_{act\_ck} \cdot N_{\mathrm{SAD}} + N_{load\_ck} \cdot N_{step}} \\ &= \frac{\frac{N^2}{N_{\mathrm{PE}}} \cdot [1 + 8\ \log_2(\omega+1)]}{\frac{N^2}{N_{\mathrm{PE}}} \cdot [1 + 8\ \log_2(\omega+1)] + (N_{\mathrm{PE}} + 2) \cdot \log_2(\omega+1)} \end{aligned} \tag{9}$$

where N_{act_ck} and N_{load_ck} are the time of active operations (in clock cycle) and data preloading (in clock cycle), and N_{step} is search steps, all in one block interval. Hence

$$OP_{\mathrm{TSS}} = N_{\mathrm{PE}} \cdot X \cdot EFF_{\mathrm{TSS}}. \tag{10}$$

Combining the preceding three equations, we obtain

$$N_{\mathrm{PE}} = \frac{x' - \sqrt{(x')^2 - 4\ \log_2(\omega+1) \cdot OP_{\mathrm{TSS}} \cdot K}}{2K \cdot \log_2(\omega+1)} \tag{11}$$

where $x' = X - 2\ \log_2(\omega+1) \cdot K$. The value of N_{PE} is smaller than 256 for CCIR-601 and CIF format pictures. Hence, we can use multiple one-column architectures to save chip area. The data flow controller in this architecture is complicated and needs more chip area than that of exhaustive-search because of the irregular data loading.

Now let us consider the worst case situation of the Type B buffer. In each search step, we need to evaluate eight or nine candidate locations and they are aligned in three rows. If the search step size is fairly large ($>N$), these data blocks do not overlap. In this case, the Type B buffer that holds the entire search region of a search step does not help in reducing the input data rate. For simplicity, we fix the processing interval for calculating each SAD operation, then we only consider two situations: 1) $\omega \geq 2N$, Type B buffer has the size of nine

TABLE II (*Continued.*)
(b) IMPLEMENTATION COMPLEXITY (2)

Algorithm (4)		Conjugate direction search		
Buffer	Input	A	$(N+2\omega)^2$	$f_r \cdot \frac{P_v}{N} \cdot [(N+2\omega)^2 + (\frac{P_h}{N} - 1) \cdot N \cdot (N+2\omega)]$
type	data rate	B	$N \times N$	$N^2 \cdot (2\omega+3) \cdot K$
size (bytes)	(bytes/sec)	C	$N \times N$	$N^2 \cdot (2\omega+3) \cdot K$
Estimated		sub/abs/add		$N^2 \cdot (2\omega+3) \cdot K$
complexity		compare		$(2\omega+3) \cdot K$
N_{PE}	$\frac{X' - \sqrt{(X')^2 - 4(2\omega+3) \cdot (2\omega-1) N^2 K^2}}{2K(2\omega-1)}, \quad X' = X - (2\omega-1) \cdot K$			
Algorithm (5)		Alternating pel-decimation search		
Buffer	Input	A	$(N+2\omega)^2$	$f_r \cdot \frac{P_v}{N} \cdot [(N+2\omega)^2 + (\frac{P_h}{N} - 1) \cdot N \cdot (N+2\omega)]$
type	data rate	$B^\dagger$	$\frac{(N+2\omega)}{2} \times \frac{N}{2}$	$\frac{1}{4} \cdot [(N+2\omega)^2 + 12N^2] \cdot K$
size (bytes)	(bytes/sec)	$C^\dagger$	$\frac{N}{2} \times \frac{N}{2}$	$\frac{1}{4} \cdot [N(N+2\omega) \cdot (2\omega+1) + 12N^2] \cdot K$
Estimated		sub/abs/add		$\frac{1}{4} \cdot N^2 \cdot [(2\omega+1)^2 + 12] \cdot K$
complexity		compare		$\frac{1}{4} \cdot [(2\omega+1)^2 + 12] \cdot K$
N_{PE}	$\frac{1}{4} \cdot \frac{K \cdot N^2 \cdot ((2\omega+1)^2+12)}{X}$			
Algorithm (6)		Subsampled motion field search with alternating pel-decimation patterns		
Buffer	Input	A	$(N+2\omega)^2$	$f_r \cdot \frac{P_v}{N} \cdot [(N+2\omega)^2 + (\frac{P_h}{N} - 1) \cdot N \cdot (N+2\omega)]$
type	data rate	$B^\dagger$	$\frac{(N+2\omega)}{2} \times \frac{N}{2}$	$\frac{1}{8} \cdot [(N+2\omega)^2 + 28N^2] \cdot K$
size (bytes)	(bytes/sec)	$C^\dagger$	$\frac{N}{2} \times \frac{N}{2}$	$\frac{1}{8} \cdot [N(N+2\omega)(2\omega+1) + 28N^2] \cdot K$
Estimated		sub/abs/add		$\frac{1}{8} \cdot N^2 \cdot [(2\omega+1)^2 + 28] \cdot K$
complexity		compare		$\frac{1}{8} \cdot [(2\omega+1)^2 + 28] \cdot K$
N_{PE}	$\frac{1}{8} \cdot \frac{K \cdot N^2 \cdot ((2\omega+1)^2+28)}{X}$			
Note	† External memory can be accessed using alternating pel-decimation patterns			

(b)

blocks (B_1 in Table II), and 2) $\omega < 2N$, it only needs to hold the search region of the first search step (B_2 in Table II). As the search step gets smaller, the overlapped area among nine candidate blocks becomes larger. Consequently, the required data loading rate decreases. The second search step, not the first step, is the worst case for data loading rate because the data locations of the first step are fixed and thus can be preloaded. Based on the assumption that every search step takes the same amount of computing time, the input data for the second search step is 1) $8N^2$ pels for $\omega > 4N - 1$, and 2) $[2\lceil(\omega+1)/4\rceil + N]^2$ pels, otherwise; where $\lceil y \rceil$ denotes the least integer greater than or equal to y. Because there are $\log_2(\omega+1)$ steps for each block, the data rate is roughly $8N^2 \log_2(\omega+1) \cdot K$ pels per second for $\omega > 4N - 1$, and $(\omega+N)^2 \cdot \log_2(\omega+1) \cdot K$ pels per second, otherwise. For the Type C buffer, the worst case is $8N^2$ new pels per search step, and therefore the data rate is $8N^2 \log_2(\omega+1) \cdot K$ pels per second. The expressions of all the above cases are listed in Table II.

3) Modified Log Search: For this search scheme, we need $1+6\log_2(\omega+1)$ SAD operations per block in the worst case. The number of PE operations required in one second is thus

$$OP_{MOD} = K \cdot [1 + 6\log_2(\omega+1)] \cdot N^2. \tag{12}$$

The number of stuffing clocks for data loading in each search step is the same as that of the three-step search algorithm. Therefore, its efficiency is

$$EFF_{MOD} = \frac{\frac{N^2}{N_{\text{PE}}} \cdot R}{\frac{N^2}{N_{\text{PE}}} \cdot R + (N_{\text{PE}}+2) \cdot 2\log_2(\omega+1)} \tag{13}$$

where $R = 1 + 6\log_2(\omega+1)$. Because $OP_{MOD} = N_{\text{PE}} \cdot X \cdot EFF_{MOD}$, we thus obtain

$$N_{\text{PE}} = \frac{x' - \sqrt{(x')^2 - 8\log_2(\omega+1) \cdot R \cdot N^2 \cdot K^2}}{4 \cdot K \cdot \log_2(\omega+1)} \tag{14}$$

where $x' = X - 4\log_2(\omega+1) \cdot K$. The column systolic architecture is adequate for both CIF and CCIR size pictures. Although each search step in the modified log search is broken into two substeps, it would be more convenient and timesaving to allow the buffer to hold the data needed for the entire search step rather than for the substep. Then, the analysis of buffer size and input data rate for this algorithm is similar to that for the three-step search (Table II).

4) Conjugate Direction Search: This algorithm requires $2\omega + 3$ SAD operations in a block time interval. When the search range ω becomes lager, the total amount of computation may be larger than that of the three-step algorithm. This is due to the fact that the number of calculations in this algorithm is on the $O(\omega)$ order and the three-step search is on the $O(\log_2 \omega)$ order.

Because of the sequential nature of this search algorithm, that is, the next step cannot start until the completion of the current step, the inserted stuffing cycles reduce the utilization of computational processors. In each search step, except for the first horizontal or vertical step, only one block of new data are needed. If we break the first horizontal step into three sequential steps and the first vertical step into two steps, then the size of the Type B buffer is the same as that of the Type C buffer, N^2 pels. Therefore, the data loading rate would be identical for the Type B and C buffers. It is, in the worst case, $(2\omega + 3) \cdot N^2 \cdot K$ pels per second.

Now we estimate the PE number, N_{PE}. Because there are at most $2\omega - 1$ search steps for one block, it takes $(2\omega - 1) \cdot (N_{\mathrm{PE}} + 2)$ data loading cycles to move data from the internal buffer to the processor array. The efficiency of processor array is thus

$$EFF_{CNJ} = \frac{\dfrac{N^2}{N_{\mathrm{PE}}} \cdot (2\omega + 3)}{\dfrac{N^2}{N_{\mathrm{PE}}} \cdot (2\omega + 3) + (2\omega - 1) \cdot (N_{\mathrm{PE}} + 2)} \tag{15}$$

and the total number of PE operations required in one second is

$$OP_{CNJ} = (2\omega + 3) \cdot N^2 \cdot K. \tag{16}$$

Again $OP_{CNJ} = N_{\mathrm{PE}} \cdot X \cdot EFF_{CNJ}$. Therefore, N_{PE} can be derived from the preceding two equations

$$N_{\mathrm{PE}} = \frac{x' - \sqrt{(x')^2 - 4(2\omega + 3)(2\omega - 1) \cdot N^2 \cdot K^2}}{2 \cdot K \cdot (2\omega - 1)} \tag{17}$$

where $x' = X - (2\omega - 1) \cdot K$.

5) Alternating Pixel-Decimation Search: In this technique, each decimation pattern contains 1/4 of the pels in a block. When the best candidates of all four decimation patterns are found, four $N \cdot N$ matchings are performed to find the overall best one. Therefore, it requires $(2\omega + 1)^2$ SDAD operations and four SAD operations. Because all the data inside the search region are used, other than the specific address pattern generated for moving data from the internal buffer to the processor array, the internal buffer size and the input data rate for the Type A buffer are identical to those of the exhaustive search. For the Type B and C buffers, because only 1/4 of the pels are used for one decimation pattern, we could complete one pattern search over the entire search area and then continue for the next one. In this case, both the input data rate and the buffer size become 1/4 of those of the exhaustive search as indicated in Table II.

Again the systolic array structure is used in this algorithm. The number of PE operations required in one block interval is roughly 1/4 of the exhaustive search, namely, $\frac{1}{4}N^2(2\omega + 1)^2$. The additional four SAD operations would add another $4N^2$ PE operations. If we store the best candidate of each search pattern, we may reduce the last four SAD operations down to $3N^2$ PE operations because four SDAD operations ($4 \times N^2/4$ PE operations) for those candidates have been done already. The total PE operations required per second is thus

$$OP_{ALT} = K \cdot \frac{N^2}{4} \cdot [(2\omega + 1)^2 + 12]. \tag{18}$$

No stuffing cycles are needed if we first calculate all the SDAD's and then four SAD's in the same sequence order (e.g., in A, B, C, D sequence) for every block. Then, the efficiency of the systolic structure is 100%. Because $OP_{ALT} = N_{\mathrm{PE}} \cdot X \cdot EFF_{ALT}$, we thus obtain

$$N_{\mathrm{PE}} = \frac{K \cdot \dfrac{N^2}{4} \cdot [(2\omega + 1)^2 + 12]}{X}. \tag{19}$$

The value of N_{PE} is larger than 64 for CCIR 601 pictures. Thus, the 2-D array architecture is employed.

6) Subsamples Motion Field Search with Alternating Pixel-Decimation Patterns: In this algorithm, we perform the APD search on the shaded blocks (half of the image blocks) and then four SAD operations on each white block. Therefore, on the average, we need $\frac{1}{2}\{N^2/4 \cdot [(2\omega+1)^2 + 12] + (N^2/4) \times 16\}$ PE operations for every two blocks. In other words, the total PE operations per second is approximately

$$OP_{SAM} = \frac{1}{2} \cdot K \cdot \frac{N^2}{4} \cdot [(2\omega + 1)^2 + 28]. \tag{20}$$

Then, the efficiency of systolic structure in this algorithm is the same as that of alternating pixel-decimation search. Consequently, the PE number, N_{PE}, is

$$N_{\mathrm{PE}} = \frac{\dfrac{1}{2} \cdot K \cdot \dfrac{N^2}{4} \cdot [(2\omega + 1)^2 + 28]}{X}. \tag{21}$$

Because the majority of computations are spent on the APD search, the same buffer sizes in Section III-B-5 for three types of buffers are adopted. However, the data rate is nearly 1/2 of those in the pure APD search because only a few data are needed for the white blocks. These values are listed in Table II.

IV. Chip Area and I/O Requirement

A. Chip Area Estimation

In order to obtain the more exact estimate of chip area, we have done two levels of simulations and analysis. One is the *behavioral level* and the other is the *structure level*. At the behavioral level, these algorithms are implemented by C-programs to verify their functionalities. At the structure level, the architectures of the key components in each algorithm are implemented using the Verilog hardware description language (HDL) and then we extract the area information from the Synopsys design tool. In our setup, the Synopsys tool produces an optimized gate-level description using a 0.6-μm single-poly double-metal (SPDM) standard cell library.

As discussed earlier, the search range depends on both the coding system structure and the applications (picture size and content). In a typical MPEG-2 encoder, the search range can be empirically decided by $15 + 16 \times (d - 1)$ [25], where d represents the distance between the target and the reference pictures. Hence, in the first application for encoding CCIR-601 pictures, the search range is chosen to be 47 for encoding P-pictures (distance $d = 3$). The chip area estimates for the computation kernels in various cases are listed in Table III. In this table, the meaning of *no. of PE operations, no. of PE*

TABLE III
ESTIMATED AREA OF COMPUTATION CORE FOR CCIR-601 PICTURES

items \ Algorithm	Exhaustive search	Three step search	Modified log search	APD search	SAPD search
No. of PE operation ($* 10^9$) (OP)	93.57	0.51	0.38	23.4	11.7
No. of PE node N_{PE}	935.7	5.17	4.1	234	117
Chosen N_{PE}	1024	8	8	256	128
Architecture Efficiency (EFF)	100%	96%	91%	100%	100%
PE Speed requirement (ns)	11	16	19	11	11
Area of PE node (Gate counts)	439	419	419	439	439
Area of ADD node (Gate counts)	260	252	252	260	260
Area of CMP node (Gate counts)	627*	257	257	627*	437*

*: multiple input comparator

TABLE IV
ESTIMATED AREA OF THE ENTIRE CHIP FOR CCIR-601 PICTURES

Items		Area \ Algorithm	Exhaustive search	Three step search	Modified log search	APD search	SAPD search
Computation core		Gate count	466.8K	3.9K	3.9K	121.3K	60.8K
		Area(mm^2)	259.1	2.2	2.2	67.3	33.7
Internal buffer	Type A	Size(bytes)	12100	12100+256	12100+256	12100+256	12100+256
		Area(mm^2)	39.1	40.1	40.1	40.1	40.1
	Type B	Size(bytes)	1760	2304+256	2304+256	440+256	440+256
		Area(mm^2)	5.6	8.2	8.2	2.3	2.3
	Type C	Size(bytes)	256	256+256	256+256	64+256	64+256
		Area(mm^2)	0.9	1.8	1.8	1.2	1.2
Address mapper		Gate count	10.4K	6.1K	6.1K	6.1K	6.1K
		Area(mm^2)	5.8	3.4	3.4	3.4	3.4
Chip area	Type A	Area(mm^2)	304	45.6	45.6	110.7	77.1
	Type B	Area(mm^2)	270.5	13.8	13.8	73	39.4
	Type C	Area(mm^2)	265.8	7.4	7.4	71.9	38.3

nodes, and *Architecture Efficiency* are defined in Section III-B as OP, N_{PE}, and EFF. The item *Chosen* N_{PE} comes from rounding the number N_{PE} up to the nearest integer of multiples of 16 (or 8) that can fit into the chosen array architecture. The speed requirement of PE node is obtained by dividing the number of PE operations (OP) by the *Chosen PE* entry. The areas of PE, ADD, and CMP are provided by the Synopsys tool under the clock rate given in the *Speed* entry.

In this design, we choose a two-port internal buffer to increase the PE utilization efficiency. The buffer size and access time requirement are determined by the chosen system architecture. However, the two-port memory module is not included in our ASIC library. Hence, an area estimation model of two-port memory proposed by Chang [26] is adopted to generate the entries in Table IV. When the chosen N_{PE} is larger than the block size, the 2-D systolic structure (Fig. 5) is then used. In the 2-D structure, the current block data can be preloaded into each PE; therefore, the current block buffer can be eliminated. It cannot be eliminated in the 1-D structure. But in either case, we always need the reference block buffer (the Type A, B, or C buffer), which is often much bigger.

A list of areas of the critical elements in various block-matching algorithms is shown in Table IV. At the end of this table, the total chip area, A_{total} specified by (4), is the combination of the computation kernel, the internal buffer, and the data mapper. It is interesting to see that the area of the internal buffer may be larger than that of the computation core. For easy comparison, the total area using different types of buffers are listed. The systolic architecture may be an inadequate choice for the conjugate direction search

TABLE V
ESTIMATED AREA OF COMPUTATION CORE FOR CIF PICTURES WITH SEARCH RANGE = 7 PELS

items \ Algorithm	Exhaustive search	Three step search	Modified log search	Conjugate direction search	APD search	SPAD search
No. of PE operation (*10^6) (OP)	228.1	25.3	19.3	17.2	60.1	32.1
No of PE node N_{PE}	5.7	0.6	0.5	0.4	1.5	0.8
Chosen N_{PE}	8	1	1	1	2	1
Architecture Efficiency (EFF)	100%	100%	100%	100%	100%	100%
PE Speed requirement (ns)	35.1	41.7	50	62.5	33.3	31.3
Area of PE node (Gate counts)	411	411	411	411	411	411
Area of AM node (Gate counts)	497	497	497	497	497	497

TABLE VI
ESTIMATED AREA OF THE ENTIRE CHIP FOR CIF PICTURES WITH SEARCH RANGE = 7 PELS

Items		Area \ Algorithm	Exhaustive search	Three step search	Modified log search	Conjugate direction search	APD search	SAPD search
Computation core		Gate count	3785	908	908	908	1319	908
		Area(mm^2)	2.1	0.5	0.5	0.5	0.73	0.5
Internal buffer	Type A	Size	900+256	900+256	900+256	900+256	900+256	900+256
		Area(mm^2)	3.8	3.8	3.8	3.8	3.8	3.8
	Type B	Size	480+256	576+256	576+256	256+256	120+256	120+256
		Area(mm^2)	2.5	2.9	2.9	1.82	1.4	1.4
	Type C	Size	256+256	256+256	256+256	256+256	64+256	64+256
		Area(mm^2)	1.82	1.82	1.82	1.82	1.25	1.25
Address mapper		Gate count	870	1574	1574	800	966	1254
		Area(mm^2)	0.48	0.87	0.87	0.44	0.54	0.7
Chip area	Type A	Area(mm^2)	6.38	5.17	5.17	4.74	5.07	5
	Type B	Area(mm^2)	5.08	4.27	4.27	2.76	2.67	2.6
	Type C	Area(mm^2)	4.4	3.19	3.19	2.76	2.52	2.45

algorithm because its pipeline efficiency is lower than 50%. From Table IV, we find that the chip area of the full-search algorithm is approximately ten times larger than that of the other algorithms for CCIR 601 pictures. If the chip area is our only concern, the three-step search and modified-log search have about the same chip area and seem to be the preferred choices. Although Type B or C buffers require smaller chip area, they demand a higher I/O bandwidth (to be discussed in the next section), we may be forced to chose the Type A buffer configuration, which has the advantages of a smaller I/O bandwidth and a simpler address generator.

In the second design, we estimate chip area for a smaller picture (CIF format) and slow motion application. Since only I-picture and P-picture are used in low-resolution video coder (H.261), two search ranges are tested: 7 and 15. Tables V and VII show the areas of the key elements in the computational kernel for CIF pictures at two search ranges. Because one PE is sufficient for fast search algorithms, their efficiency is 100%. The estimated chip areas are listed in Tables VI and VIII. Because the I/O bandwidth limitation is not severe in this case, Type B or C buffers could be reasonable choices in this case. We find that the conjugate direction search has a somewhat lower chip area. However, the conjugate direction search often has the lowest image quality (Section V). Therefore, the other fast searches are also good candidates. For a search range of 7 pels, the chip areas for all algorithms are quite close.

From the above results, we find that the buffer portion is the dominant factor in chip area for the sequential searches particularly for small size pictures. In contrast, the area of the computation core dominates the entire chip area for the exhaustive type of searches (including APD and SAPD) particularly for large size pictures. Therefore, if the picture size

TABLE VII
ESTIMATED AREA OF COMPUTATION CORE FOR CIF PICTURES WITH SEARCH RANGE = 15 PELS

Algorithm / items	Exhaustive search	Three step search	Modified log search	Conjugate direction search	APD search	SPAD search
No. of PE operation $(*10^6)$ (OP)	974.2	33.5	25.3	33.5	246.6	125.3
No of PE node N_{PE}	24.4	0.8	0.6	0.8	6.16	3.1
Chosen N_{PE}	32	1	1	1	8	4
Architecture Efficiency (EFF)	100%	100%	100%	100%	100%	100%
PE Speed requirement (ns)	32.8	31.25	41.7	31.25	32.5	30.8
Area of PE node (Gate counts)	411	411	411	411	411	411
Area of AM node (Gate counts)	497	497	497	497	497	497

TABLE VIII
ESTIMATED AREA OF THE ENTIRE CHIP FOR CIF PICTURES WITH SEARCH RANGE = 15 PELS

Items		Area / Algorithm	Exhaustive search	Three step search	Modified log search	Conjugate direction search	APD search	SAPD search
Computation core		Gate count	13649	908	908	908	3785	2141
		Area(mm^2)	7.6	0.5	0.5	0.5	2.1	1.2
Internal buffer	Type A	Size	2116+256	2116+256	2116+256	2116+256	2116+256	2116+256
		Area(mm^2)	7.58	7.58	7.58	7.58	7.58	7.58
	Type B	Size	736+256	1024+256	1024+256	256+256	184+256	184+256
		Area(mm^2)	3.27	4.17	4.17	1.83	1.61	1.61
	Type C	Size	256+256	256+256	256+256	256+256	64+256	64+256
		Area(mm^2)	1.83	1.83	1.83	1.83	1.25	1.25
Address mapper		Gate count	900	1620	1620	800	1020	1300
		Area(mm^2)	0.5	0.9	0.9	0.44	0.57	0.73
Chip area	Type A	Area(mm^2)	15.68	8.98	8.98	8.52	10.25	9.51
	Type B	Area(mm^2)	11.37	5.57	5.57	2.77	4.28	3.54
	Type C	Area(mm^2)	9.93	2.23	2.23	2.77	3.92	3.18

and/or search range are huge, the three-step search and the modified-log search have about the same chip area and are the preferred choices. We like to emphasize again that our estimates of chip area are rough and the architecture used here is not tuned to a particular algorithm, although it may happen to be a better implementation of certain algorithms. Therefore, the analysis here provides only the global picture and overall tendency rather than the accurate and final specifications. For the same reason, we did not perform the same analysis on many different size pictures and other search ranges since we can already see the advantages and disadvantages of these search algorithms through some representative cases.

B. Chip I/O Configurations

The number of I/O pads is one major factor in chip fabrication cost. There are roughly three types of I/O pins: $PAD_{\rm MEM}$ is the bus width connected to the external memory, $PAD_{\rm control}$ and $PAD_{\rm power}$ are the pads for control signal and power supply. Although the values of $PAD_{\rm control}$ and $PAD_{\rm power}$ may depend on the system architecture, there are no simple rules to estimate them. Often, they do not vary very much. (It was reported [11] that they are around 28.) We now only look into the bandwidth requirement due to input data. There are two approaches in calculating the I/O bandwidth requirement. We could assume a minimum external memory access time (decided by the available DRAM, say) and then calculate the minimum bus width, $PAD_{\rm MEM}$. Or, we first assume the $PAD_{\rm MEM}$ value, and then calculate the maximum allowable memory access time. In Table IX, the latter approach is taken and we assume that $PAD_{\rm MEM}$ equals to W. The necessary input data speeds in various cases are calculated based upon the discussions in Section III-B. For example, for the CCIR picture application, if the Type B buffer

TABLE IX
EXTERNAL MEMORY ACCESS TIME REQUIREMENT

Algorithm / M type		Full search	Three step search	Modified log search	Conjugate direction search	APD search	SAPD search
CCIR format (ω=47)	A	1.551W	1.551W	1.551W	1.551W	1.551W	1.551W
	B	0.255W	0.322W	0.322W	0.124W	0.814W	1.28W
	C	0.018W	0.251W	0.251W	0.124W	0.073W	0.142W
CIF format (ω=15)	A	39.52W	39.52W	39.52W	39.52W	39.52W	39.52W
	B	14.92W	13.7W	13.7W	3.74W	24.34W	27.2W
	C	1.38W	3.58W	3.58W	3.74W	4.88W	8.42W
CIF format (ω=7)	A	63.25W	63.25W	63.25W	63.25W	63.25W	63.25W
	B	35.07W	26.3W	26.3W	7.25W	31.79W	31.3W
	C	4.38W	5.14W	5.14W	7.25W	12.29W	17.58W

*: the unit is ns/bit,
W: the bus width for external memory

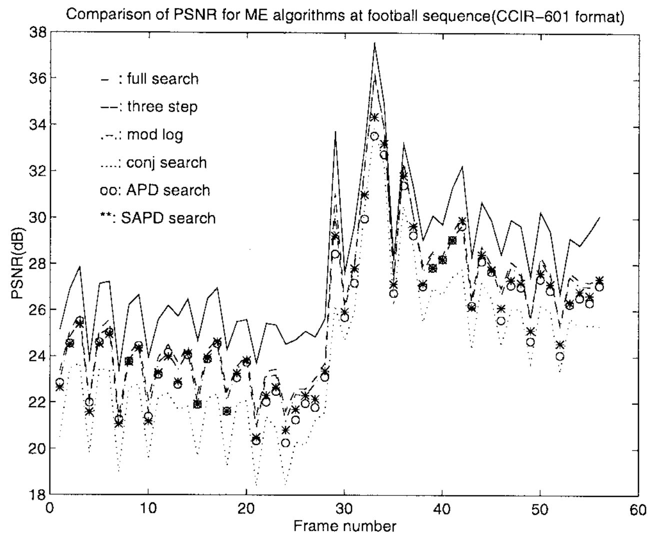

Fig. 10. PSNR performance of motion estimation algorithms on the CCIR *Football* sequence.

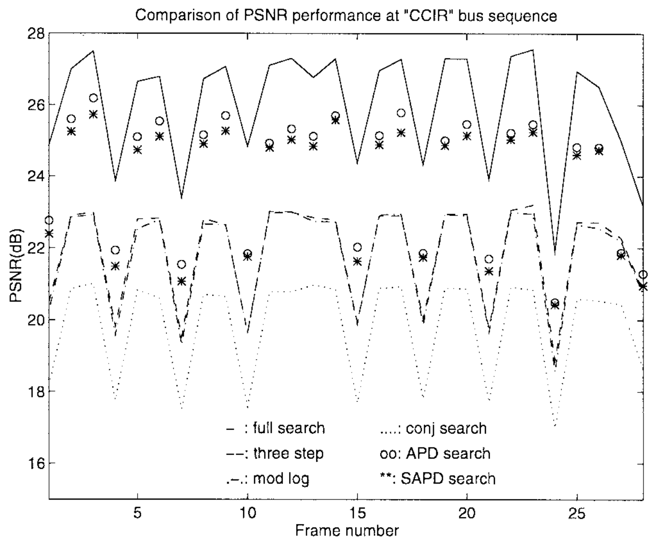

Fig. 11. PSNR performance of motion estimation algorithms for the CCIR *Bus* sequence.

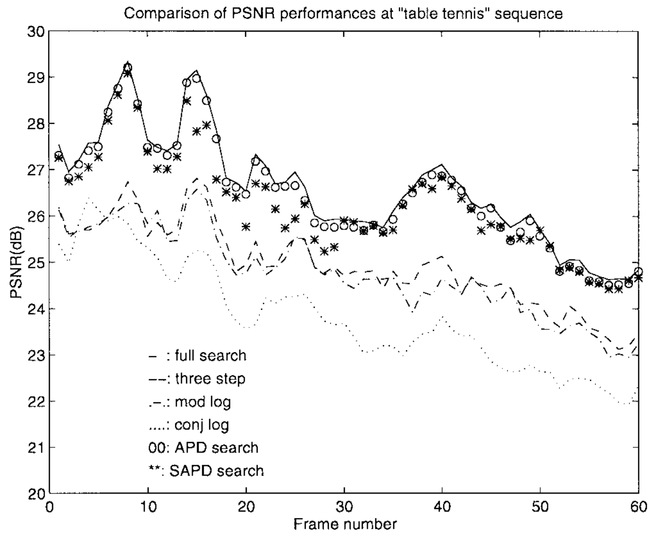

Fig. 12. PSNR performance of motion estimation algorithms on the CIF *Table Tennis* sequence, search range = 7 pels.

is chosen and the external memory bus width is 64 ($W = 64$), this table tells us that the external memory access time must be less than $0.255 \times 64 = 16$ ns for the full (exhaustive) search. A larger access time implies an easier situation that either we could find a slower speed DRAM to meet our requirement or we could reduce the memory bus width (smaller W). As one may expect, the Type A buffer is preferred at the cost of a larger internal buffer. Practically, if the ordinary low cost DRAM is used as the external memory with an access time of 60 ns and the bus width (W) is around 60 too, then the entries less than $1\,W$ in Table IX are not acceptable. That is, the Type A buffer is nearly the only choice for CCIR pictures with a search range of 47. On the other hand, if W is greater than 20, all three types of buffers can be used for CIF pictures with a search range of 15 or less. One additional remark is that in the case that the motion estimator is a part of a video encoder chip, it may not own completely the I/O pins, but its bandwidth requirements usually have a strong impact on the entire chip—the motion estimator is often the most demanding unit on data access.

V. PICTURE QUALITY

Since different block-matching algorithms are used, their image qualities are not identical. Although peak signal-to-noise ratio (PSNR) may not be a good measure for the subjective image quality, it can still be used as an indicator for quality comparison. The PSNR is defined as the ratio of the peak signal power (255^2) to the mean square motion estimation errors. Several sequences have been tested. Limited by space, only three of them are reported here. Another measure of the effectiveness of a motion estimation algorithm is the number of bits necessary to transmit the estimation errors. For simplicity, we calculate only the first-order entropy [3].

Figs. 10 and 11 show the motion estimation errors of the CCIR 601 image sequences *Football* and *Bus*. Only the processed picture frames are displayed in the figures. In this simulation, the search range is 47 for P-pictures and 15 for B-pictures. Figs. 12 and 13 show the results of two 10-frames/s

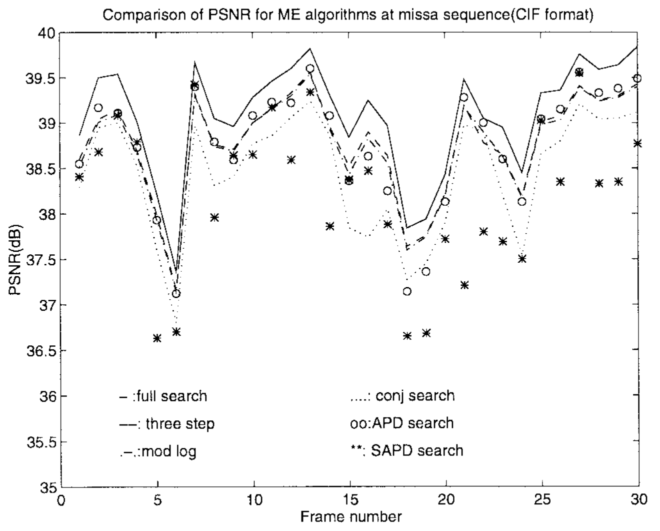

Fig. 13. PSNR performance of motion estimation algorithms on the CIF *Miss American* sequence, search range = 15 pels.

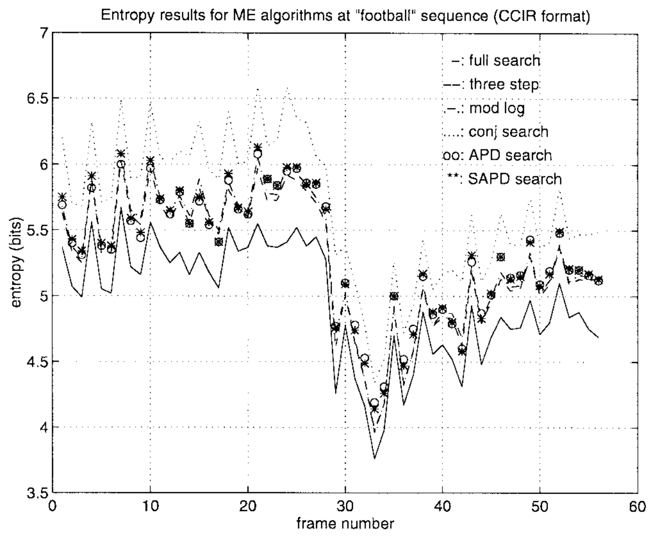

Fig. 14. Entropy performance of motion estimation algorithms on the CCIR *Football* sequence.

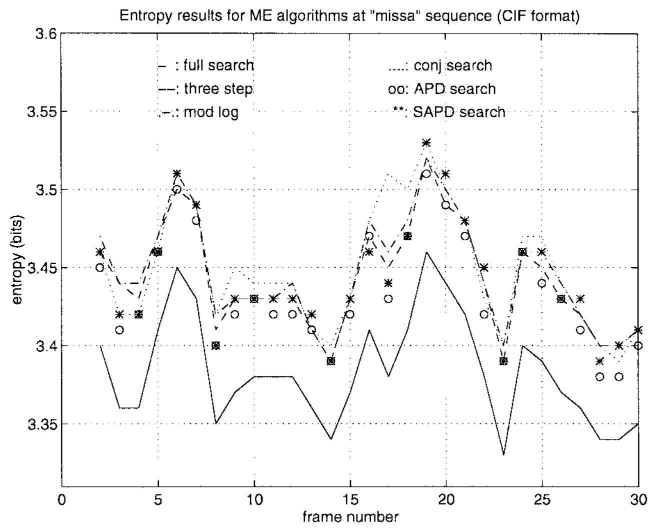

Fig. 15. Entropy performance of motion estimation algorithms on the CIF *Miss American* sequence, search range = 15 pels.

CIF image sequences *Table Tennis* and *Miss America* with a search range of 7 and 15, respectively. Except for the first frame, the rest are all P-pictures. It is clear that the full search algorithm outperforms all the other algorithms. The three-step search and the modified log search are lower by roughly 1 dB in PSNR. Their PSNR values in all four sequences are very close to each other. Similar trends are shown in the entropy results, Figs. 14 and 15. In general, the pixel-decimation technique (APD) has a better performance on slowly moving pictures (such as *Table Tennis* and *Bus*) but has a poorer performance on fast moving pictures (such as *Football*). The conjugate direction search has a PSNR quite a bit lower than that of the other search algorithms. Hence, unless there is a significant advantage in hardware cost, the conjugate direction search is not preferred from the image quality viewpoint. One may note that in order to match the video coding standards in which the block is 16 $\times$ 16, the so-called *subblock* in the original SAPD [24] is now 16 $\times$ 16 rather than 8 $\times$ 8. The larger subblock size slightly reduces its performance. If the picture quality is our major concern, the exhaustive search is the best choice. However, for large size pictures and/or large search ranges, the three-step and the modified-log searches have a much lower hardware cost and only a somewhat lower picture quality.

VI. Conclusions

The purpose of this study is not to propose a VLSI architecture for implementing a specific BMA, but to evaluate various block-matching algorithms from the viewpoints of both VLSI design and compression efficiency. A procedure is suggested to assist VLSI designers to choose a good block-matching algorithm adequate for their particular applications. Our assessment on BMA in this paper is based on silicon area, I/O requirement, and image quality. A universal systolic arrays structure is used to realize all the BMA candidates. A distinct feature in our study is to look into the effect of different sizes of the on-chip memory. Although we did not complete the layout of each realization, the key elements in the hardware have been implemented using Verilog language and their silicon areas are extracted with the help of Synopsys tool based on a 0.6-μm SPDM standard cell library.

Examples of applications at CIF and CCIR-601 picture resolutions are examined. In summary, we found that the relative performance in chip area and I/O bandwidth between various algorithms is strongly picture size- and search range-dependent. For small pictures (CIF, for example) and slow motion (small search range), all the BMA's under consideration are on a par. However, for larger picture sizes (CCIR-601) and fast motion, certain fast search algorithms have the advantage of a significantly smaller chip area. For a specific algorithm, one may tune the hardware structure to achieve an even more economical design. Nevertheless, we have conducted a comprehensive study on estimating the complexity of various motion estimation algorithms, their chip area, data bandwidth, and image quality. This analysis should be able to provide useful guidelines to the system designers in choosing a suitable high-level algorithm for VLSI implementation.

REFERENCES

[1] K. Kucukcaker and A. C. Parker, “A methodology and design tools to support system-level VLSI design,” Univ. Southern California, Dept. Electrical Eng.-Syst., Tech. Rep., June 1994.

[2] H. G. Musmann, P. Pirsch, and H.-J. Grallert, “Advances in picture coding,” *Proc. IEEE,* vol. 73, pp. 523–548, Apr. 1985.

[3] A. N. Netravali and B. G. Haskell, *Digital Pictures: Representation, Compression and Standards.* New York: Plenum, 1995.

[4] A. M. Tekalp, *Digital Video Processing.* Upper Saddle River, NJ: Prentice Hall, 1995.

[5] H.-M. Hang and Y.-M. Chou, “Motion estimation for image sequence compression,” in *Handbook of Visual Communications,* H.-M. Hang and J. W. Woods, Eds. San Diego, CA: Academic, 1995.

[6] K. R. Rao and J. J. Hwang, *Techniques and Standards for Image, Video, and Audio Coding.* Upper Saddle River, NJ: Prentice Hall, 1996.

[7] T. Akari *et al.,* “Video DSP architecture for MPEG2 codec,” in *Proc. ICASSP’94.* IEEE Press, 1994, vol. 2, pp. 417–420.

[8] T. Inoue *et al.,* “Programmable vision processor/control for flexible implementation of current and future image compression standards,” *IEEE Micro,* vol. 12, pp. 33–39, Oct. 1992.

[9] J. Goodenough *et al.,* “A general purpose, single chip video signal processing (VSP) architecture for image processing, coding and computer vision,” in *IEE Colloquium on Parallel Architectures for Image Processing,* 1994, pp. 1/1–1/4.

[10] P. Pirsch, N. Demassieux, and W. Gehrke, “VLSI architectures for video compression—A survey,” *Proc. IEEE,* vol. 83, no. 2, pp. 220–246, Feb. 1995.

[11] K. M. Yang, M. T. Sun, and L. Wu, “A family of VLSI design for the motion compensation block-matching algorithm,” *IEEE Trans. Circuits Syst.,* vol. 36, pp. 269–277, Oct. 1989.

[12] L. De Vos and M. Stegherr, “Parameterizable VLSI architectures for the full-search block-matching algorithm,” *IEEE Trans. Circuits Syst.,* vol. 36, pp. 1309–1316, Oct. 1989.

[13] S. H. Nam, J. S. Beak, and M. K. Lee, “Flexible VLSI architecture of full search motion estimation for video applications,” *IEEE Trans. Consumer Electron.,* vol. 40, May 1994.

[14] A. Costa *et al.,* “A VLSI architecture for hierarchical motion estimation,” *IEEE Trans. Consumer Electron.,* vol. 41, May 1995.

[15] H.-K. Jung *et al.,* “A VLSI architecture for the alternative subsampling-based block matching algorithm,” *IEEE Trans. Consumer Electron.,* vol. 41, pp. 231–238, May. 1995.

[16] L. De Vos and M. Schöbinger, “Efficient architecture of a programmable block matching processor,” in *Intel. Conf. Application-Specific Array Processors,* Oct. 1993, pp. 560–571.

[17] Y. S. Jehng, L. G. Chen, and T. D. Chiuh, “An efficient and simple VLSI architecture for motion estimation algorithms,” *IEEE Trans. Signal Processing,* vol. 41, pp. 889–899, Feb. 1993.

[18] H.-D. Lin *et al.,* “A programmable motion estimator for a class of hierarchical algorithms,” in *VLSI Signal Processing VIII.* New York: IEEE Press, 1995.

[19] T. Komarek and P. Pirsch, “Array architectures for block matching algorithms,” *IEEE Trans. Circuits Syst.,* vol. 36, pp. 269–277, Oct. 1989.

[20] R. Aravind *et al.,* “Image and video standards,” in *Handbook of Visual Communications,* H.-M. Hang and J. W. Woods, Eds. San Diego, CA: Academic, 1995.

[21] T. Koga *et al.,* “Motion-compensated interframe coding for video conferencing,” in *Proc. Nat. Telecommunications Conf.,* New Orleans, LA, Nov. 1981, pp. G5.3.1–G5.3.5.

[22] S. Kappagantula and K. R. Rao, “Motion compensated predictive interframe coding,” *IEEE Trans. Commun.* vol. COM-33, pp. 1011–1015, Sept. 1985.

[23] R. Srinivasan and K. R. Rao, “Predictive coding based on efficient motion estimation,” *IEEE Trans. Commun.,* vol. COM-33, pp. 888–896, Sept. 1985.

[24] B. Liu and A. Zaccarin, “New fast algorithms for the estimation of block motion vectors,” *IEEE Trans. Circuits Syst. Video Technol.,* vol. 3, pp. 148–157, Apr. 1993.

[25] A. Puri, R. Aravind, and B. Haskell, “Adaptive frame/field motion compensated video coding,” *Signal Processing: Image Commun.,* vol. 5, pp. 39–58, 1993.

[26] T. S. Chang, “On-chip memory module designs for video signal processing,” Master thesis, Institute of Electronics Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., June 1995.

Sheu-Chih Cheng received the B.S. degree in electronics engineering from National Taiwan Industrial Technology, Taipei, Taiwan, in 1989 and the M.S. degree from National Chiao Tung University, Hsinchu, Taiwan, in 1991. He is currently working toward the Ph.D. degree in electronics engineering at National Chiao Tung University.

His research interests are video coding and VLSI design for signal processing.

Hsueh-Ming Hang (S79–M’80–SM’91) received the B.S. and M.S. degrees from National Chiao Tung University, Hsinchu, Taiwan, in 1978 and 1980, respectively, and the Ph.D. degree in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1984.

From 1984 to 1991, he was with AT&T Bell Laboratories, Holmdel, NJ. He joined the Electronics Engineering Department of National Chiao Tung University, Hsinchu, Taiwan, in December 1991.

Dr. Hang was a Conference Co-Chair of the Symposium on Visual Communications and Image Processing (VCIP), 1993, and the Program Chair of the same conference in 1995. He guest co-edited two *Optical Engineering* special issues on Visual Communications and Image Processing in July 1991 and July 1993. He was an Associate Editor of IEEE TRANSACTIONS ON IMAGE PROCESSING from 1992 to 1994 and a co-editor of the book *Handbook of Visual Communications* (Academic Press, 1995). He is currently an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY and an Editor of *Journal of Visual Communication and Image Representation* (Academic Press). He is a member of Sigma Xi.