An Enhanced Three Step Search Motion Estimation Method and its VLSI Architecture

Prasad Lakamsani, Bing Zeng, and Ming Liou
Department of Electrical and Electronic Engineering
The Hong Kong University of Science and Technology

Abstract - Recent studies show that the motion vector distribution within the search window is center-biased. Based on this fact, we propose in this paper an Enhanced version of the well-known three step search hierarchical block motion estimation method, specifically aiming towards the H.261 standards, and show that its performance is much better than the original, without any direct or hidden costs. We also propose a simple VLSI architecture to implement the proposed method on silicon with the minimum number of gates and extend it to implement the regular three step search method.

1. Introduction

Recent progress in the field of communications based applications like integrated services digital network (ISDN) videophone and videoconferencing systems are mainly concentrated on minimizing the cost and size of the codec equipment because low cost of final product is the most essential part of development at the current age of technology. One way to achieve this goal is to reduce the amount of data to be transmitted by making use of some kind of compression techniques, like motion-compensated hybrid coding that has been widely in use among a number of compression techniques, and was adopted by several international standards. This is due to the fact that the block matching motion estimation/compensation reduces the temporal redundancy within frames, which is a necessity to reduce the bit-rate. However, even though the performance of the well-known full-search (FS) method undoubtedly is the best, it has very high computational complexity because motion estimation is done on the block by block basis. Since the implementation complexity mostly depends on the complexity of the algorithm itself, the FS method needs a huge number of gates for its implementation.

Several fast-search algorithms have been developed, such as three step search (TSS) [6], 2-D log search (2-D Log) [2], conjugate direction search (CDS) [5], one-at-a-time search (OTS) [5] to name a few. Researchers made several attempts to use these fast algorithms, especially TSS - as each and every pixel of the search window can be reached during the search, for almost all types of applications: videophone, MPEG, digital TV, HDTV - though standards and quality requirements are quite different from one another. However, our belief is that better solutions can be achieved, both in performance and gate count, if we can come up with algorithms that are specific for a particular application rather than a group of applications. In this paper, we detail our attempt to enhance the performance of TSS without increasing the complexity of the algorithm (we call it as enhanced TSS, ETSS) aimed especially towards H.261 standards.

While attempting to develop a new block motion estimation algorithm one has also to think of the feasibility to realize it in hardware using today's technology as well. In general, fast algorithm implementation indeed is much simpler than FS and requires rather less number of gates. There were several attempts made to reduce the gate count for the above mentioned implementations, especially a great many architectures were proposed to implement TSS on silicon: parallel, tree, and array architectures to mention a few. Keeping these in mind, we propose a simple architecture to implement the ETSS method. We will show that a saving of more than 60% can be achieved using our architecture, as far as the gate count is concerned. We also show that the same architecture can be extended to implement other hierarchical block matching algorithms as well.

Rest of this paper is organized as follows: We describe the idea behind TSS method and its limitations in section 2 and propose our enhanced method, which is aimed towards the H.261 standards, that overcomes the limitation of the TSS in section 3. We discuss the proposed architecture in detail in section 4. We extend our discussion further to implement TSS with the same architecture in section 5 followed by conclusions in section 6.
2. Three step search method and its limitation

Basic idea behind block matching methods is to match each block in the current frame $A$ with each of the candidate blocks in the previous frame $B$ that has a block size $N \times N$, search range shifted from $-d$ to $+d$ in each direction. In general, mean absolute difference ($MAD$) between blocks is considered to be the matching criteria because of its simplicity in calculation. For an $N \times N$ block, with its left pixel located at co-ordinate $(m, n)$ in the current frame $A$, MAD is calculated by

$$MAD(i, j) = \sum_{m=1}^{N} \sum_{n=1}^{N} |A(m, n) - B(m + i, n + j)|$$

where $-d \leq i, j \leq d$

and the block with minimum distortion is considered for prediction.

Unlike full-search, the main problem with any of the fast block matching methods is their inability to locate the global minimum precisely. It is very easy to be trapped into local minima in the first step as only few points are to be considered. Increasing the total number of points automatically increase the chances of going very near to the global minimum. In general, that also increases the complexity of the implementation. Recent studies showed that motion vector distribution is concentrated at the center of the search window [4]. So, chances of getting closer to the global minima are increased, and hence a better performance is expected, if more blocks are to be employed near the center. But, the first step of TSS has no special consideration to the center of the search window as it was designed to spread the search points uniformly throughout the window.

3. Enhanced three step search (ETSS) method

Keeping the importance of the above mentioned fact in mind we have modified the first step of TSS in such a way that the concentration is more at the center: by intelligently arranging the available nine search points without increasing their number. Instead of spreading the search points all over the search window, we have tried to pull them as close to the center point as possible in the search window. In this way, we were able to reduce the distance between center point and surrounding points significantly. In other words, our search pattern is more biased towards the center of the search window due to which the performance is expected to be far better than TSS.

With ETSS, as shown in figure 2, it is impossible to reach all of the 225 points in the search window of $15 \times 15$ (i.e., -7 to +7). Even then, the amount of improvement is quite significant for H.261 applications where motion is small. Performance improvement can be achieved further by decreasing the size of the search window to $11 \times 11$ (i.e., -5 to +5), instead of the normal search window of $15 \times 15$, in which case it is possible to reach any point in the search window.

Our approach differs from TSS in the first step since it is very crucial and the most important step in hierarchical block matching methods. Once the best match is found in the first step, a local search will be carried out in second and third steps.

Simulations have been carried out for a block size of $16 \times 16$ with window size 7 on standard slow motion sequences, Miss America and Salesman for e.g., to compare the performance of the proposed enhanced method with that of others. Results of the missA sequence are shown in figure 3. As we can see, ETSS outperformed TSS at every point and has performed better than NTSS at times which has used more number of search points than TSS or ETSS.

With these results, we have substantiated our claim of better performance achievement with ETSS by using the same number of search points as in TSS.

4. Proposed architecture for ETSS

The proposed ETSS is aimed for H.261 applications where motion is expected to be very small. Due to this, we have focused on reducing the gate count in this proposed architecture. We have decided to use only the minimum number of processing elements since the throughput requirement is not high for H.261.

When dealing with huge amounts of data, like in signal and image processing applications, one way of escaping from the i/o problems is to use on-chip memory, which in general, needs large amounts of extra gates, which implies that the size of the chip area will increase very much. However, this is not true in every case. Though extra gates are needed, the benefits that associated with on-chip memory are quite attractive and so we have considered using it. We have saved both reference and search windows on-chip. Due to this, we were able to use the minimum number of processing elements (PE's).
6. Conclusions

In this paper, we presented a new methodology for block motion estimation, especially for video telephone and video conferencing applications where motion is very small. Though the number of search points is the same as in TSS, far better performance has been achieved with our approach. On contrary to the general belief that gate count increases quite rapidly with the use of on-chip memory, we have obtained a simple and direct structure for the proposed method with on-chip memory. Our architecture turns out to be very economical owing to its small number of gates, less than 10K.

The same architecture can be used to implement other hierarchical block motion estimation methods, with very minor extensions. We have showed this by taking TSS implementation as an example. So, our architecture is more of a general architecture to implement hierarchical block matching methods for H.261 applications rather than one used specifically for ETSS.

References


Fig. 1: Three step search

Fig. 2: Each of the three steps of ETSS

Fig. 3: Performance comparison
Table 1: Comparison of other architectures

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of PEs</td>
<td>48</td>
<td>9</td>
<td>64</td>
<td>3</td>
</tr>
<tr>
<td>External data accesses / block matching</td>
<td>2376</td>
<td>1280</td>
<td>788</td>
<td>1024</td>
</tr>
<tr>
<td>Clock cycles</td>
<td>230</td>
<td>794</td>
<td>1024</td>
<td>2304</td>
</tr>
<tr>
<td>Throughput (in K)</td>
<td>173.91</td>
<td>50.4</td>
<td>39.06</td>
<td>17.36</td>
</tr>
<tr>
<td>Gate equiv. (in K)</td>
<td>32.9</td>
<td>36.8</td>
<td>136.3</td>
<td>17.19</td>
</tr>
</tbody>
</table>

Fig. 4: Data Flow with 3PEs

Fig. 5: Memory Interleaving

Fig. 6: Time Scheduling with Memory Interleaving