Linear Hashtable Motion Estimation Algorithm for Distributed Video Processing

Yunsong Wu[1,2], Graham Megson[2]

[1] Jiangxi Science & Technology Normal University
Nanchang, China
[2] School of Systems Engineering, Reading University
Reading, UK
{sir02yw, g.m.megson}@rdg.ac.uk

Abstract. This paper presents a parallel Linear Hashtable Motion Estimation Algorithm (LHMEA). Most parallel video compression algorithms focus on Group of Picture (GOP). Based on LHMEA we proposed earlier [1][2], we developed a parallel motion estimation algorithm focus inside of frame. We divide each reference frames into equally sized regions. These regions are going to be processed in parallel to increase the encoding speed significantly. The theory and practice speed up of parallel LHMEA according to the number of PCs in the cluster are compared and discussed. Motion Vectors (MV) are generated from the first-pass LHMEA and used as predictors for second-pass Hexagonal Search (HEXBS) motion estimation, which only searches a small number of Macroblocks (MBs). We evaluated distributed parallel implementation of LHMEA of TPA for real time video compression.

Keywords: Parallel Algorithm, Distributed Computing, Distributed Video Coding, Linear Hashtable, Motion Estimation

1 Introduction

In this paper, a parallel Linear Hashtable Motion Estimation Algorithm (LHMEA) for the Two-Pass Algorithm (TPA) constituted by LHMEA and Hexagonal Search (HEXBS) to predict motion vectors for inter-coding [1] is proposed. The objective of the motion estimation scheme is to achieve good quality video with very low computational time and low transmission rate. It is hard to find software solutions that efficiently code high-quality video in real-time or faster. We propose and evaluate distributed parallel implementations of the LHMEA of TPA on clusters of workstations for real time video compression as test. It discusses how distributed video coding on load balanced multiprocessor systems can help, especially on motion estimation. The software platform used for these is the Parallel Virtual Machines (PVM) programming model and C respectively. The effect of load balancing for improved performance will also be discussed. This paper is only concerned with the Block Matching Algorithms (BMA), which is widely used in MPEG2, MPEG4, and H.263. In BMA, each block of the current video frame is compared to blocks in

reference frame in the vicinity of its corresponding position. It is highly desired to speed up the process of compression without introducing serious distortion. The HEXBS is a widely accepted fast motion estimation algorithm [2]. The Linear Algorithm and Hexagonal Search Based Two-Pass Algorithm (LAHSBTPA) previously proposed has an improvement over the HEXBS on compression rate, PSNR and compression time. In the last 20 years, many fast algorithms have been proposed to reduce the exhaustive checking of candidate Motion Vectors (MV). Such as Two Level Search (TS), Two Dimensional Logarithmic Search (DLS) and Subsample Search (SS) [3], the Three-Step Search (TSS), Four-Step Search (4SS) [4], Block-Based Gradient Descent Search (BBGDS) [5], and Diamond Search (DS) [6], [7] algorithms. A very interesting method called HEXBS has been proposed by Zhu, Lin, and Chau [8]. The fast BMA increases the search speed by taking the nature of most real-world sequences into account while also maintain a prediction quality comparable to Full Search. Most algorithms suffer from being easily trapped in a non-optimum solution. LHMEA based TPA sorts out this problem. Normally video encoders are very effective reducing the size of the video stream, but the processing cost is very high for high quality video sequences. Although there are hardware video encoders available, they have severe restrictions (resolution, coding options, etc). A more flexible choice is to use distributed parallel implementations. Processing video with high performance distributed computing has great potential and good future, but the studies in these fields mainly concentrated on Group of Pictures (GOP) separation.

To take advantage of the potential processing power of distributed computing, we use distributed programming techniques based on message passing. We have used PVM because there are free implementations available and it is a widely accepted standard.

Various image and video compression algorithms use parallel processing. Approaches used can largely be divided into four areas. The first is the use of special purpose architectures designed specially for image and video compression. An example of this is the use of an array of DSP chips to implement a version of MPEG. The second approach is the use of VLSI techniques. The third approach is algorithm driven, in which the structure of the compression algorithm describes the architecture, e.g. pyramid algorithms. The fourth approach is the implementation of algorithms on high performance parallel computers.

The TPA which we have proposed has achieved best result in all the algorithms in the survey. To further improve the result and speed, the most suitable and easiest way is using parallel algorithm to implement the algorithm on high performance parallel computers. In the first-pass coding of TPA, LHMEA is employed to search all Macroblocks (MB) in the picture. Because LHMEA is based on a linear algorithm, which fully utilizes optimized computer's structure based on addition, so it is easy to be paralleled. Meanwhile HEXBS is one of the best motion estimation methods to date. The new method proposed in this paper achieves the best results so far among all the algorithms investigated on compression rate, time and PSNR.

Contributions from this paper are:

1. The TPA achieves the best results among all investigated BMA algorithms.
2. Improved Hashtable is used in video encoding.
3. The parallel algorithm improves LHMEA of TPA. It implements and shows better compression speed, and fair compression rate and PSNR than original TPA.

4. Work load balancing algorithm is implemented in the hashtable image encoding process.

The rest of the paper is organized as follows. Section 2 continues with an introduction to improved LHMEA and TPA and gives experimental result showing TPA's advantage over other algorithms. The proposed parallel algorithm and its implementation for LHMEA are introduced in Section 3. Experimental results showing paralleled hashtable compared with the original are also included in Section3. The paper concludes in Section 4 with some remarks and discussions about the proposed scheme.

1 Sequential and Parallel Implementation of Linear Hashtable Motion Estimation Algorithm (LHMEA)

Our method attempts to predict the motion vectors using linear algorithm.[1][2] It uses hashtable method into video compression. After investigating of most traditional and on-the-edge motion estimation methods, we use latest optimization criterion and prediction search method. Spatially MBs' information is used to generate the best motion vectors[8]. We designed a vector hashtable lookup matching algorithm which is more efficient method to perform an exhaustive search: it considers every macroblock in the search window. This block-matching algorithm calculates each block to set up a hashtable. It is a dictionary in which keys are mapped to array positions by a hash function. We try to find as few variables as possible to represent the whole macroblock. Through some preprocessing steps, "integral projections" are calculated for each macroblock. These projections are different according to different algorithm. The aim of these algorithms is to find best projection function. The algorithms we present here has 2 projections. One of them is the massive projection, which is a scalar denoting the sum of all pixels in the macroblock. It is also DC coefficient of macroblock. The other is A of Y=Ax+B (y is luminance, x is location.) Each of these projections is mathematically related to the error metric. Under certain conditions, the value of the projection indicates whether or not the candidate macroblock will do better than best-so-far match.

2.1 Sequential Implementation of LHMEA

The followings are the pseudo code, theory time, practical time calculation of linear hashtable motion estimation algorithm. The algorithm is used in pre-computation part of in MPEG codec and implemented in both sequential and parallel ways. In the program, we try to use polynomial approximation to get such y=mx+c; y is luminance value of all pixels; x is the location of pixel in macroblocks. The way of scan y is from left to right, from top to button. Coefficients m and c are what we are looking for. As shown in the figure below.

In this function y=f(x), x will be from 0 to 255 in a 16*16 pixels macroblock, y=f(x)=mx+c.

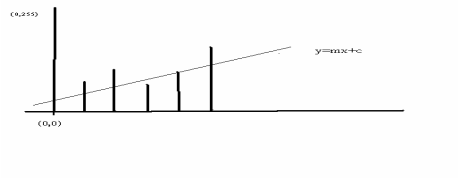

Fig. 1. Linear algorithm for discrete algorithm

$$m = \frac{N * \sum_{i=0}^{N} (x_i * y_i) - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} y_i}{N * \sum_{i=0}^{N} x_i^{\,2} - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} x_i} \quad (\mathbf{1})$$

$$c = \frac{\sum_{i=0}^{N} y_i * \sum_{i=0}^{N} x_i^{\,2} - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} x_i * y_i}{N * \sum_{i=0}^{N} x_i^{\,2} - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} x_i} \quad (\mathbf{2})$$

Here we state the pseudo code to calculate the hashtable function: The function to implement the algorithm is encapsulated in MyMotionSearchPreComputation Mpeg-Frame *frame)

Sequential Code:

Step 1: if ((psearchAlg == VECTOR_HASH || psearchAlg == HEX_VECTOR_HASH || psearchAlg == HEX) && (frame->type == I_FRAME || frame->type == P_FRAME))
Step 2: EnterTimeCount(0)
Step 3, Paral: if (IsSetUpHashTablePVM) { call PVM Motion Search PreComputation;}
else{
Step 4: if(HashTableSearchType) InitMHashTable();
Step 5: for (y = 0; y < Fsize_y - 16; y++) {
Step 6: for (x = 0; x < Fsize_x - 16; x++)
Step 7: { call different hashtable setup functions }
Step 8:if (use HashTable) { add M,C,X,Y into hashtable }}}}
Step 9: LeaveTimeCount(0);

MB is transferred by hash function to hash coefficients, M,C,X,Y generated are added into hashtable.

In previous research methods, when people try to find a block that best matches a predefined block in the current frame, matching was performed by SAD (calculating difference between current block and reference block). In Linear Hashtable Motion Estimation Algorithm (LHMEA), we only need to compare two coefficients of two blocks. In current existing methods, the MB moves inside a search window centered on the position of the current block in the current frame. In LHMEA, the coefficients move inside hashtable to find matched blocks. If coefficients are powerful enough to

hold enough information of MB, motion estimators should be accurate. So LHMEA increases speed and accuracy to a large extent.

From the pseudo code above, we can get calculation time in theory:

The precomputation complexity is the function (3)

$$T_{seq}(n_{seq}, s_{seq}, \phi_{seq}) = n_{seq} \times s_{seq} \times \phi_{seq} \qquad (\mathbf{3})$$

The variables inside the function are

1. n_{seq} : reference frame number, which is also number of I, P frames

2. s_{seq} : frame size, which in the program is

$$(width\_frame) \times (length\_frame) \qquad (\mathbf{4})$$

3. ϕ_{seq} : the complexity to calculate the hash function per macroblock, which will be explained later.

So the complexity of the linear hashtable motion estimation algorithm depends on the three variables.

To demonstrate the complexity of calculation, the following example is given:

The video sequence used in the experimentation is three YUV (352x240 pixels) test sequence, which is known as Flower Garden sequence. There are 150 frames in the original sequences, which sub sampled to the 4:1:1 format in the YUV color space. The video sequence was divided into several sections (GOPs), each of which contained 15 frames to be compressed and a reference frame. A frame pattern of IBBPBBIBBPBBPBB was used. The average time is defined as the overall execution time of the group, including the I/O time, the computation time and the communications time. The motion vector search algorithm used is the LHMEA based TPA and produces integer pixel motion vectors.

We calculate it in details here to demonstrate how it is working.

n_{seq} =50 out of 150 frames.

$s_{seq} = (width\_frame - MB\_size) \times (length\_frame - MB\_size)$ =75264.

According to the complexity of calculate Macroblock,

ϕ_{seq} depends on the hash function calculation method.

For the coefficients m and c we mentioned earlier:

$$m = \frac{N * \sum_{i=0}^{N} (x_i * y_i) - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} y_i}{N * \sum_{i=0}^{N} x_i^{\,2} - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} x_i} \qquad (\mathbf{1})$$

$$c = \frac{\sum_{i=0}^{N} y_i * \sum_{i=0}^{N} x_i{}^2 - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} x_i * y_i}{N * \sum_{i=0}^{N} x_i{}^2 - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} x_i} \qquad (2)$$

In the C codec, We only calculate $\sum_{i=0}^{N}(x_i * y_i)$ and $\sum_{i=0}^{N} y_i$, because for a 16x16 macroblock, $\sum_{i=0}^{N} x_i$, $\sum_{i=0}^{N} x^2{}_i$, N, $N * \sum_{i=0}^{N} x_i^2 - \sum_{i=0}^{N} x_i * \sum_{i=0}^{N} x_i$ can be pre-calculated before calling the function.

In the codec, pseudo code decides the complexity of ϕ_{seq} is as following:

so ϕ_{seq} =16*16*[1 (*)+5 (+)]+ 4 (*)+2 (+)

```
for(iy=0;iy<MB_size;iy++){
    for(ix=0;ix<MB_size;ix++){
        temp1= frame->ref_y[y+iy][x+ix];
        sum_yi += temp1;
        sum_xiyi += count* temp1;
        count ++;   } }
        (*pnowBuildTable)[y][x].B = 0.125*sum_yi;
        (*pnowBuildTable)[y][x].A
        =(12*sum_xiyi-6*(total_size+1)*sum_yi)>>10;
```

In this example total sequential time in theory is

$T_{seq}(n, s, \phi)$

$$= n_{seq} \times s_{seq}(M^2{}_{frame\_dimension}) \times \phi_{seq}(N^2{}_{MB\_dimension} \times \gamma_{hashfunctioncomplexity}) \qquad (5)$$

$$= n_{seq} \times M^2{}_{frame\_dimension} \times N^2{}_{MB\_dimension} \times \gamma_{hashfunctioncomplexity} \qquad (6)$$

=50[(Fsize_x-MB_size)*(Fsize_y-MB_size)]*{16*16*[1 (*)+5 (+)]+ 4 (*)+2 (+)}*
=978432000 (*)+4824422400 (+)

Practical sequential time counting: $T_{seq}(n, s, \phi)$ =7.2763(s)

2.2 Parallel Implementation of LHMEA

In the parallel implementation, to parallelize an encoder, we divide each reference frames (which can be I or P frame) into equally sized regions. Current frames are also divided into non-overlapped regions. These regions are going to be processed in parallel to increase the encoding speed significantly. Each region is divided into non-overlapping range blocks. Each region will be sent to corresponding slaves and generates its own hashtable table. The slave will be alive until encoding finishes. Slaves will generate its own hashtable and Motion Vectors table, sending MVs table back to the master. However, there is an upper limit on the number of PEs that can be used due to the limited spatial resolution of a video sequence. Also a massive spatial

parallel algorithm usually needs to tolerate a relatively large communication over-head. In our approach of spatial parallelism, load balancing was implemented to ensure an equal distribution of the frame data among the processors.

Here we state the pseudo code to calculate the hashtable function in parallel. The function to implement the algorithm is encapsulated in PreComputation()

Parallel Code:
Input: part of reference frame from master
Output: part of hashtable

```
Step 1:   rcode=pvm_upkint (FrameData,Fsize_X*(rows),1); /*Get Data from Master*/
Step 2:   /*Give Data from buffer to Reference Frame, */
          for(i=0;i< Fsize_X *(rows);i++)
          {prevFrame.ref_y[tempy][tempx] = FrameData[i];}
Step 3:   For (i=0;i< rows;i++)
Step 4:         for (k=0;k< Fsize_x-16; k++){
Step 5:             for(iy=0;iy<16;iy++){
Step 6:                 for(ix=0;ix<16;ix++)
                    {calculate sum_xi*yi and sum_yi for each Pixel;}}
Step 7:                 calculate M and C for each Pixel;}}}
```

The structure of the algorithm can be demonstrated in the figure 2.
The reference frame are divided into several parts,

$$rows = (width\_frame - width\_MB) / N\_PCs + searchwindows$$

are sent to clients.

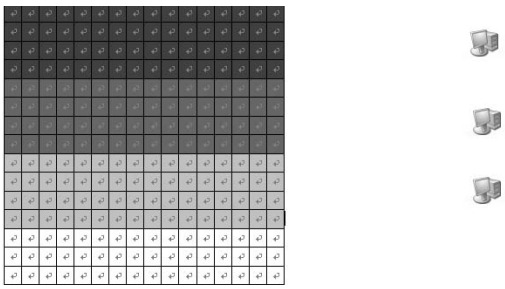

Fig. 2. The Parallel Structure of Hashtable

From the pseudo code above, we can get calculation time in theory:
The precomputation complexity is function

$$T_{paral}(n_{paral}, s_{paral}, \phi_{paral}) = n_{paral} \times s_{paral} \times \phi_{paral} \quad (7)$$

The variables inside the function have similar meaning as in sequential function

1, $n_{paral} = n_{seq}$

2, s_{paral} : frame size, which is whole frame divided by Number_PCs

$$((width\_frame - width\_MB) / N\_PCs + searchwindows) * (length\_frame) \quad (8)$$

3, per macroblock. pseudo code decides the complexity of $\phi_{paral} = \phi_{seq}$

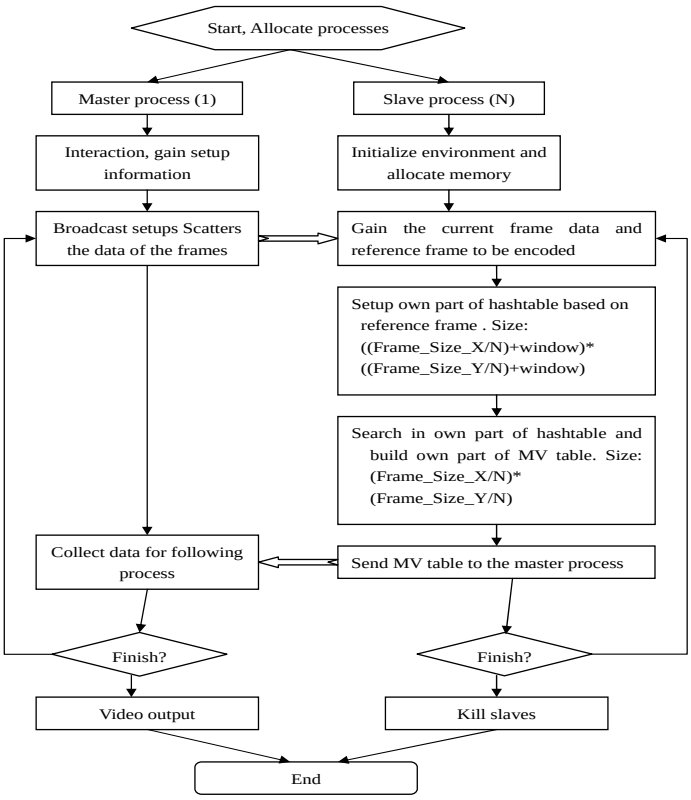

Fig. 3. Process of parallel LHMEA setup

Using the same example sequences of frames and number of slaves equal to 4, if we use 2 slaves, each slave will get *rows=352/2 + search window=216* for each. If we use 4 slaves, each slave will get *rows=352/4 + search window*, 108,128,128,108 for each. We use biggest one to calculate totally time for slaves.

In this case:

When the number of PCs=2:

$$T_{paral}(n_{paral}, s_{paral}, \phi_{paral}) = n_{paral} \times s_{paral} \times \phi_{paral}$$

$$= n_{paral} \times s_{paral}(M^2{}_{frame\_dimension}) \times \phi_{paral}(N^2{}_{MB\_dimension} \times \gamma_{hashfunctioncomplexity}) \quad (9)$$

$$= n_{paral} \times \frac{M^2{}_{frame\_dimension}}{N\_PCs} \times N^2{}_{MB\_dimension} \times \gamma_{hashfunctioncomplexity} \quad (10)$$

=50[(rows-MB_size)*(Fsize_x-MB_size)]*{16*16[1(*)+5(+)]+4(*)+4(+)}*
50[124*336]*[260(*)+1284(+)]*
= 541632000 (*)+2674828800 (+)

Speedup: $\tau = \dfrac{T_{seq}(n_{seq}, s_{seq}, \phi_{seq})}{T_{paral}(n_{paral}, s_{paral}, \phi_{paral})}$ =

$$\frac{978432000\,(*) + 4824422400\,(+)}{541632000\,(*) + 2674828800\,(+)} = 1.8065$$

Practical sequential time counting: $T_{seq}(n, s, \phi)$ =7.2763(s)

The figure 4& 5 below are Time Spent, Actual Speed Up, Theory Speed Up comparison for parallel LHMEA based on the 150 Flower Garden Sequences. PSNR and compression rate remain the same as sequential algorithm [1][2].

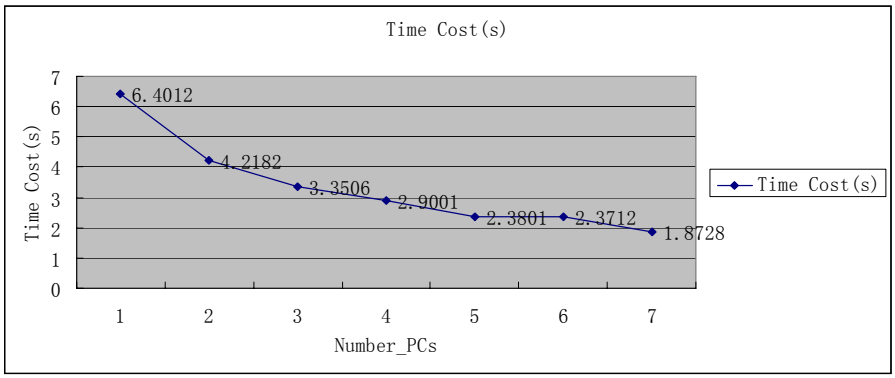

Fig. 4. Time cost decrease with Number of PCs

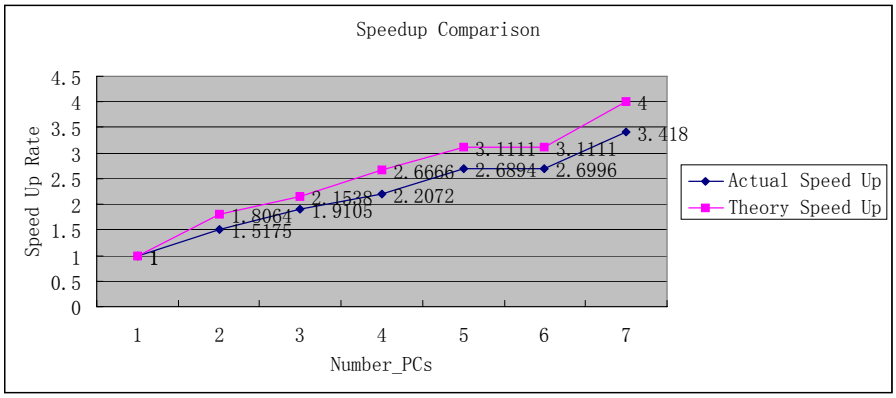

Fig. 5. Actual Speed Up, Theory Speed Up comparison

In theory, the speedup should be in linear increasing with number of PCs. The reason why it does not match a linear model is that we are not sending exact

Frame/Number_PCs data to slaves, instead, we send Fsize_y/Number_PCs plus search windows size rows data to slaves. Also it is limited by resolution of images. More data ($2 \times window\_size \times Frame\_width$) will be calculated than the original frame. In theory, the larger number of PCs, the more redundant data. The curve of speedup-Number PCs will have less descent when the PCs number increases.

Time cost also depends on the speed of PCs. We use a network of workstations comprises similar workstations linked together by a common network e.g. Ethernet. When CPU clock is counted, the faster the PC, the less time it takes.

3 Conclusion

In the paper, a parallel Linear Hashtable Motion Estimation Algorithm (LHMEA) and Hexagonal Search Based Two-Pass Algorithm (TPA) in video compression is proposed based on the LHMEA. The hashtable is used in video compression and implemented with parallel computing in motion estimation. The algorithm searches in the hashtable to find the motion estimator in-stead of by full search algorithm in whole frame. Then the LHMEA was implemented in parallel algorithm. The speedup of paralleled LHMEA is compared to the original sequential LHMEA. The parallel video coding is implemented inside frame rather than between frames. The key point in the method is to find suitable hash function to produce the hashtable.

References

1. Yunsong Wu, Graham Megson, "Linear Predicted Hexagonal Search Algorithm with Moments", ICIC 2005, Part I, Springer LNCS 3644, pp. 136 – 145, (2005).
2. Yunsong Wu, Megson G, "Two-pass hexagonal algorithm with improved hashtable structure for motion estimation Pro-ceedings." IEEE Conference on Advanced Video and Signal Based Surveillance, pp. 564 – 569, (2005).
3. Ce Zhu, Xiao Lin, Lappui Chau, and Lai-Man Po, "Enhanced Hexagonal Search for Fast Block Motion Estimation", IEEE Trans on Circuits and Systems for Video Technology, Vol. 14, No. 10, (Oct 2004)
4. Qiang Peng; Yulin Zhao, "Study on parallel approach in H.26L video encoder", PDCAT'2003. Proc of the Fourth International Conference, p:834 – 837 Aug. (2003)
5. K. Shen, L. A. Rowe, E. J. Delp. "A Parallel Implementation of an MPEGI Encoder: Faster Than Real-Time!". Proc of the SPIE - The International Society for Optical Engineering, ~01.2419p, p:407-418.
6. M. Ribeiro, 0. Sinnen, L. Sousa. "MPEG-4 Natural Video Parallel Implementation on a Cluster". 12th (RECPAD2002), Portugal, June (2002).
7. H. Ning, J. T. Li and S. X. Lin. "A Study of Parallelism in MPEG-4 Video Encoder", Journal of Computer Engineering and Applications, Vol 38, pp.9-12, July, (2002)
8. Alexis M. Tourapis, Oscar C. Au, Ming L. Liou, "Predictive Motion Vector Field Adaptive Search Technique (PMVFAST) Enhancing Block Based Motion Estimation", Proc Visual Communications and Image Processing, San Jose, CA, January (2001)