Optimal Scheduling Algorithms for Communication Constrained Parallel Processing

D. Turgay Altılar and Yakup Paker

Dept. of Computer Science, Queen Mary, University of London
Mile End Road, E1 4NS, London, United Kingdom
{altilar, paker}@dcs.qmul.ac.uk

Abstract. With the advent of digital TV and interactive multimedia over broadband networks, the need for high performance computing for broadcasting is stronger than ever. Processing a digital video sequence requires considerable computing. One of the ways to cope with the demands of video processing in real-time, we believe, is parallel processing. Scheduling plays an important role in parallel processing especially for video processing applications which are usually bounded by the data bandwidth of the transmission medium. Although periodic real-time scheduling algorithms have been under research for more than a decade, scheduling for continuous data streams and impact of scheduling on communication performance are still unexplored. In this paper we examine periodic real-time scheduling assuming that the application is communication constrained where input and output data sizes are not equal.

1 Introduction

The parallel video processing scheduling system studied here assumes a real-time processing with substantial amount of periodic data input and output. Input data for such a real-time system consists of a number of video sequences that naturally possess continuity and periodicity features. Continuity and periodicity of the input leads one to define predictable and periodic scheduling schemes for data independent algorithms. Performance of a scheduling scheme relies upon both the system architecture and the application. Architectural and algorithmic properties enable to define relations among the number of processors, required I/O time, and processing time. I/O bandwidth, processor power, and data transmission time, could be considered as architectural properties. Properties of the algorithm indicate the requirements of an application such as the need of consecutive frames for some computation. In this paper, two scheduling and data partitioning schemes for parallel video processing system are defined by optimising the utilisation of first I/O channels and then processors. Although it is stated that the goal of high performance computing is to minimise the response time rather than utilising processors or increasing throughput [1], we have concentrated both on utilisation side and response time. In the literature, there are a number of cost models such as the ones defined in [1],[2],[3],[4],[5] and [6]. We defined scheduling and data partitioning schemes that can work together. The

B. Monien and R. Feldmann (Eds.): Euro-Par 2002, LNCS 2400, pp. 197–206.
© Springer-Verlag Berlin Heidelberg 2002

parameters for defined schemes reflect the features of the chosen parallel system architecture and algorithm class. The defined schemes could be used for finding the optimal number of processors and partitions to work on for each scheduling model. In an other way around, system requirements could also be computed for a specific application, which enables us to build the parallel processing system.

The target parallel architecture is a client-server based system having a point-to-point communication between the server and client processors, which are required to implement Single Program Multiple Data (SPMD) type of programming. A typical hardware configuration comprises a server processor, a frame buffer and a number of client processors connected via a high speed I/O bus and signal bus. Video data transfer occurs over the high speed I/O bus between clients and the frame buffer. The frame buffer is a specially developed memory to save video streams. Since the frame buffer can provide only one connection at a time, any access to the frame buffer should be under the control of an authority, the server, to provide the mutual exclusion. The server is responsible to initialise clients, to partition data, to sent data addresses to clients to read and write, and to act as arbiter of the high speed I/O bus. No communication or data transfer exists between client processors.

Digital video processing algorithms can be classified under two groups considering their dependency on the processing of the previous frames. If an algorithm runs over consecutive frames independently we call it *stream based processing* [7] which is not considered in this paper. If an algorithm requires the output from the previous frame of a stream, the computation of a frame can proceed when the previous frame is processed. We call this mode *frame by frame processing*. In order to run a frame by frame computation in parallel, a frame can be split into tiles to be distributed to client processors. These tiles are processed and then collected by the server to re-compose the single processed frame.

Parallel Recursive (PR) and Parallel Interlaced (PI) scheduling algorithms are suggested in this paper for parallel video processing applications that require the output from the preceding frame to start with a new one. Video input/output is periodic. A new frame appears for every 40 ms for a PAL sequence. Input and output size are unequal for many of the video processing algorithms . Such as in mixing two sequences outputs size is roughly one third of the input.

The rest of the paper is organised as follows: Section 2 introduces the mathematical modelling and relevant definitions that are used in analysis of scheduling models. Equal data partitioning scenarios are discussed and analysed in Section 3. Scheduling for unequal input and output are investigated and new algorithms are proposed and analysed in Section 4. Section 5 compares all the introduced methods via a case study. Paper ends with conclusions and further research.

2 Mathematical Modeling and Definitions

Read and write times can be best defined as a linear function of input data size and bus characteristics. The linear functions include a constant value, p for read and s for write, which identifies the cost of overhead. These constant costs are

considered as initialisation costs due to the system (latency) and/or due to the algorithm (data structure initialisations). Data transfer cost is proportional to another constant q for read and t for write. Computation time is accepted as proportional to the data input size. r is computational cost per unit data. It is important to note that r is not a complexity term. d_i indicates the partition of the data in percentage to sent i^{th} processor. Throughout the following derivations only input data size is taken as a variable. Consistant with the existing literature and cost models referred in the introduction, the developed a cost model includes first degree equations for cost analysis although numeric solutions always exist for higher degree equations. For the i^{th} processor read R_i, compute C_i and write W_i times can be expressed as follows where the sum of all d_i is 1;

$$R_i = p + qd_i,\ C_i = rd_i,\ W_i = w + td_i \tag{1}$$

Sending data from frame buffer to client processors, processing in parallel and receiving processed data from all of the available client processors constitutes a cycle. Processing of a single frame finishes by the end of a cycle. Since our intention is to overlap compute time of a processor with I/O times of the others, starting point of the analysis is always an equation between read, compute and write times. In order to make a comparative analysis the response time, T_{cycle}, is essential. Also note that T_{cycle} provides a means to compute speed up.

3 Equal Data Partitioning

Partitioning data in equal sizes is the simplest and standard way of data partitioning to provide load balancing. Data is partitioned into N equal sizes to be dispatched to N processors. While processors compute their part of data, they obviously leave the I/O bus free. Utilisation of the I/O Bus depends on these idle durations. Whenever a processor starts computation another one starts a read. This continues in the same manner for other processors until the very first one finishes processing its data and becomes ready to write its output via the I/O bus. One could envisage a scenario that the computation time of the first processor is equal to the sum of the read time of others so that no I/O wait time is lost for the first processor. Therefore, the maximum number of processors is determined by the number of read time slots available for other processors within computation time of the first processor. In order to ensure the bus becomes free when the first processor completes computation, the compute time for the first processor must be equal to or greater than the sum of all reads. Similarly for the second processor's computation time can be defined as the sum of read times of the successor processors and the write time of the first one. If one continues for the subsequent processors, it is easy to see that compute time for i^{th} processor must be greater than or equal to the sum of read times of the successor processors and the sum of write times of the predecessor processors. Assuming that N is the number of processors, to achieve the full utilisation of data bus

computation time should equal to sum of communication times:

$$C_i = \sum_{k=1}^{i-1} W_k + \sum_{j=i+1}^{N} R_j \tag{2}$$

By substituting definitions of R, W and C given in Eq.1 in Eq.2 and solving the produced quadratic function the positive root can be found as follows:

$$N = \left((p-q) + \sqrt{(p-q)^2 + 4pr} \right) / 2p \tag{3}$$

The lower bound of N is the optimal value for N for the utilisation of I/O bus. Moreover, the cycle time (T_{cycle}), posing another constraint to be met in real time video processing can be computed as the sum of all writes and reads, i.e. $T_{\text{cycle}} = 2\left(\lfloor N \rfloor p + q\right)$.

3.1 Equal Data Partitioning with Unequal I/O

However, when input and output data sizes (or cost factors) become different equal partitioning can not provide the best solution. There can two cases of unequal input and output data transfer: input data takes longer to transfer than output or vice-versa.

Write time greater than read time. The first case is for a generic class of the algorithms with larger output data size than input such as rendering a 3D scene. Rendering synthetic images, the data size of 3D modelling parameters (polygons) to construct the image is less than the rendered scene (pixels). If processors receive equal amount of data they all produce output after a computation time which is almost the same for each of them. As writing output data takes longer than reading input data, the successor processor waits the predecessor to finish its writing back. Although the load balancing is perfect, i.e. each processor spends the same amount of time for computation, I/O channel is not fully utilised. In Fig.1a, L_2, L_3, and L_4 indicate the time that processors spend while waiting to write back to the frame buffer. We keep the same approach as we analyse the equal input and output case: computation time should overlap data transfer time (either read or write) of the other processors. It can be seen in Fig.1a that computation time of the first processor can be made equal to the read time of the rest of the processors. For the second processor however, the derivation introduces a new period called L for the idle duration of the processor as W_1 R_2 (Note that all read times are equal as well as write times). Therefore the difference between read and write time produces an idle duration for the successor processor. The latency for the second processor is $L2 = W1 - R2$. The sum of all idle durations for all client processors is $L_{\text{total}} = \left(N^2 - N\right) L_2/2$ As shown in Fig.1a, although I/O channel is fully utilised client processors are not. Moreover, the cycle time is extended by the idle time of the last client processor taken part in the computation. The overall parallel computation cycle time is: $T_{cycle} = N(R + W)$.

Read time greater than write time. The second generic case (Fig.1b) occurs when writing takes less time than reading data. Consider motion estimation of MPEG video compression which reads a core block (called "macro block" in MPEG terminology) of a by a pixels from the current frame to be matched with neighbouring blocks of previous frame within a domain of $(2b+1)(2b+1)$ pixels centred on the macro block where b could be up to $16a$ [8]. However the output is only a motion vector determining the direction of the macro block. The second step of the derivation introduces a new duration called I for the idle duration of the I/O bus. The difference between read and write time produces an idle duration for the I/O bus which can be given as I=R-W. As a processor finishes writing earlier than the start of writing of its successor there is no queuing effect. The sum of idle durations of the I/O bus, it I_T, is proportional to the number processors, $I_T = (N-1)I$, and T_{cycle} becomes: $T_{cycle} = (2N-1)R + W$.

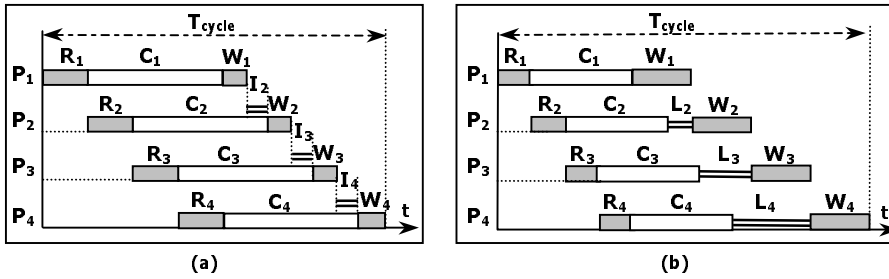

Fig. 1. Equal data partitioning with (a) write time greater than read time and (b) read time greater than write time

4 Scheduling for Unequal I/O

We have shown in Section 3 that equal data partitioning for equal load distribution does not always utilise the I/O channel and/or the processors fully. The following figures (Fig.2a, Fig.2b and Fig.2c) show the three possible solutions based on two new partitioning approaches. The main objective is to maximise I/O Bus utilisation since we assume applications are bounded by data transfer. We also assume that the algorithm is data independent and data can be partitioned and distributed in arbitrary sizes.

4.1 PR Scheduling and Data Partitioning

Parallel Recursive (PR) data partitioning and scheduling method exploits the computation duration of a processor for its successor to proceed with its I/O. As the successor processor starts computation, the next one can start its I/O. This basic approach can be recursively applied until the compute time becomes

not long enough for read-compute-write sequence of the successor processor. Although utilisation of the I/O channel would be high, and cycle time would be better than the equal data partitioning (PE) method, it suffers from the under utilisation of processors. Recursive structure of the scheduling and partitioning provides a repetitive pattern for all the processors. Since subsequent processors exploit duration between read and write times of the first processor, cycle time is determined by the first processor. The computation time of the first processor which leaves I/O bus idle is used by the second one. The same relationship exists between the second processor and the third one and so on. Although read time is greater than write time in Fig.2a, the following equations are also valid for the other two scenarios in which (i) write time is greater than read time and (ii) write time and read time are equal. The first processor dominates the cycle time. One can define compute time considering Fig 6 for N processors: $C_i = R_{i+1} + C_{i+1} + W_{i+1}$ Since the sum of all reads and writes is equal to $T_{cycle} - C_N$. T_{cycle} can be derived as follows in terms of system constants:

$$T_{cycle} = \frac{N(p+s)+q+t}{1-\left(\frac{r}{q+r+t}\right)^N} - \frac{(p+s)r}{q+t} \tag{4}$$

Data partitions can be calculated as follows using the relation between two consecutive data partitions:

$$d_{N-m} = \frac{a^{N-m}}{r}\left(\frac{N(p+s)+q+t}{1-a^N} - \frac{br}{a-1}\right) + b\frac{a^{N-m}-1}{a-1} \tag{5}$$

where $a = r/(q+r+t)$ and $b = -(p+s)/(q+r+t)$

The number of processors to maximise the utilisation of I/O channel is also a question worth considering. The recursive structure of the model leaves smaller task for a processor than its predecessor. After a number of successive iterative steps the compute time of a processor will not be sufficient for its successor for read and write as the overall working time becomes smaller for the successor processors. This constraint poses a limit for the number of processors. In the case of computing data partition size for an insufficient slot the computed data size would be negative. N can be computed numerically via the following inequality:

$$\frac{Na^N(p+s)+q+t}{1-a^N} \rangle \frac{br}{a-1} + q + t \tag{6}$$

4.2 PI Scheduling and Data Partitioning

Parallel Interlaced (PI) scheduling and data partitioning method is another proposed method to maximise the utilisation of the I/O bus. Unlike PR the basic approach is for each processor to complete its read-compute-write cycle after its predecessor but before its successor. This is the same approach that we use to analyse equal input and output. The two other possible scenarios is analysed in this section. Fig.2a and Fig.2b show the possible solutions for unequal

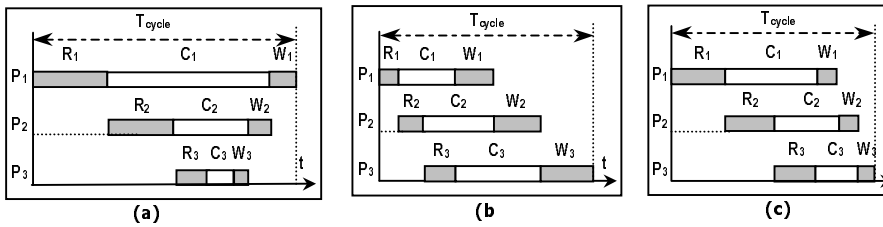

Fig. 2. Optimal data partitioning with (a) write time greater than read time and (b) read time greater than write time

input/output. For the first case given in Fig.2b, since writing requires more time than reading, computation time should increase with the processor number in order to accommodate longer writing times. Since read, compute, and write times are proportional to data size, from Fig.2b we can say that ascending read, compute and write times increases with the increasing index of processors provides full utilisation of the I/O channel for an application with longer write time than read. The second case is shown in Fig.2c where reading requires more time than writing. Thus, computation time should decrease with the increase of processor number in order to accommodate shorter writing times. A close look at Fig.2c shows that with increased processor numbers the read compute and write times are also increased. So long as the read time is longer than the write time, the difference reduces the time for the successor processor to read and compute. Although the difference between write time and read time provides an additional time for the successor processor in one case (Fig.2b), and reduces the time for the other case (Fig.2c) the compute time and response time satisfy the following equations for both of the cases:

$$T_{cycle} = C_i + \sum_{k=1}^{i} R_k + \sum_{j=i}^{N} W_j \tag{7}$$

$$C_i + W_i = R_{i+1} + C_{i+1} \tag{8}$$

One can solve these equations for d_n as follows

$$d_n = (r+t)^{n-1}(r+q)^{N-n}\left(\frac{t-q+N(s-p)}{(r+t)^N-(r+q)^N}\right) - \frac{s-p}{r-q} \tag{9}$$

Thus for a given number of processors N and systems and algorithmic constants, data partitions can be computed. We dealt with a relation between two consecutive data partitions and which allows us to derive recursively all the others. However, since the aim is high utilisation of the I/O channel, data partitions should also fulfil the constraints. These constraints derived from the relations between compute time of one processors with read and write times of the others. We are going to deal with two constraints, which could be considered as upper and lower bounds. If these two constraints, one is about d1 and the other is

about d_N, are satisfied the in-between constraints will also be satisfied. The first constraint, for the first processor, is that the sum of consecutive reads excluding the first one should be greater than or equal to the first compute time, C_1, which is a function of d_1: $d_1 \geq (p(N-1)+q)/r + q$. The second constraint, for the final processor, is that the sum of consecutive writes excluding the last one should be greater than or equal to the last compute time, C_N, which is a function of d_N: $d_1 \geq (s(N-1)+q)/r + s$. If computed data partition size is less than any of these two limits values, data transmission time will be less than the compute time which yields poor utilisation of the I/O bus and increase in cycle time.

5 Comparison of Data Partitioning Schemes

In order to compare the given three methods, PE, PR, and PI, data partitions and cycle times for a single frame (T_{cycle}) have to be computed. This comparison will indicate the shortest cycle time which is crucial for real-time video processing. On the other hand, there are constraints to be satisfied in order to utilise the I/O channel. Fig.3 and Fig.4 give a brief information about the data partitions with constraints and cycle times. Since we are dealing with a colour PAL video sequences of 576*720 pixels, 24 bit colour and 25 frames per second a single PAL frame is approximately 1.2 Mbytes and has to be processed within 40 ms.

The algorithm considered in this example is mixing which requires three video streams: two streams to mix and one alpha frame to define the layering. Initialisation for reading which includes both algorithmic and systems delay is assumed to be 3.00 ms. Initialisation duration for writing is assumed to be less than reading and is 1.20 s, i.e., p=3.00 ms and s=1.20 ms. Assuming that the bus is rated 1GBytes/sec and since three streams are required for input and one is produced for output for mixing overall read and write times are $R_{overall}$=3.6ms and $W_{overall}$=1.2ms. Therefore q=3.60ms and t=1.20 ms. Given a CPU with a clock rate of 300MHz, assume that the algorithm requires 30 cycles per pixel - which can be found either by rehearsal runs on a single CPU or analysing the machine code of the program - to compute ends with a total processing time $W_{overall}$ 120ms i.e, r=120 ms. Fig.3 and Fig.4 are produced for p=3.00 ms, q=3.60 ms, r=120 ms, s=1.20 ms, and t=1.20 ms with regard to the given analysis and derived equations. Partition percentages and cycle times per frame for equal partitioning (PE) method is given in Table.1. The first row of the table indicates cycle times for different numbers of processors. Obviously the best result is 43.00 ms for 6 processors. The last two lines of constraints for data partitions are also satisfied for 6 processors. Therefore the best overall process cycle can be declared as of 6 processors. Data partitions would be equal for processors and each processors would receive approximately 17% of the input to process. However overall processing time of 43ms does not satisfy the real-time constraint of video processing for a PAL sequence of 25 frames per second. The number of processors can be computed by Eq.3 as 6.3195. The lower bound of N is equal to 6. Therefore 6 processors give the best solution for highly utilisation of

(PE)

PARTITIONS	PROCESSORS 1	2	3	4	5	6	7
T_{cycle}	129.0	71.40	54.20	47.10	44.04	43.00	43.11
1	1.00	0.50	0.33	0.25	0.20	0.17	0.14
2		0.50	0.33	0.25	0.20	0.17	0.14
3			0.33	0.25	0.20	0.17	0.14
4				0.25	0.20	0.17	0.14
5					0.20	0.17	0.14
6						0.17	0.14
7							0.14
$d_1 \geq ...$	-	0.05	0.08	0.10	0.13	0.15	0.17
$d_N \geq ...$	-	0.02	0.03	0.04	0.05	0.06	0.07

(PR)

PARTITIONS	PROCESSORS 1	2	3	4	5	6	7
T_{cycle}	129.0	69.96	51.75	43.76	39.88	38.07	37.45
1	1.00	0.53	0.38	0.32	0.29	0.27	0.27
2		0.47	0.33	0.27	0.24	0.23	0.22
3			0.29	0.23	0.20	0.18	0.18
4				0.18	0.16	0.14	0.14
5					0.12	0.10	0.10
6						0.07	0.06
7							0.03

Fig. 3. Data partitions and cycle times for PE and PR

I/O channel. Rounding the number to its lower bound yields a deviation from the optimal solution. Fig.3 shows the results for recursive partitioning (PR) method. Best cycle time is found for 7 processors, i.e., 37.45 ms. As PR method is recursive there is no constraint due to the data size except for the fact that partitions should be positive percentages. For eight processors, the size of data partition for the eight processor is computed to be less than zero. Therefore the maximum number of processors for this case is 7. The results for interlaced partitioning is shown in Fig.4. The best overall cycle time is 36.82ms for 8 processors. However partitions given for 8 processors do not satisfy the constraints given in last two rows of the table. The first column fulfilling the constraints is for 7 processors. The overall cycle time is 36.85ms which also satisfies 40 ms maximum processing time constraint. The cycle time values for the three methods are drawn in Fig.4.

Obviously PI has the best performance, where PR comes the second and PE the third. One can see the change of slopes of the curves at different values. The point on which slope is zero indicates optimum number of processors to provide the shortest cycle time if this value satisfies the constraints as well.

6 Conclusion and Further Research

In this paper, we proposed two optimal data partitioning and scheduling algorithms, Parallel Recursive (PR) and Parallel Interlaced (PI), for real-time fram by frame processing. We also provide analysis and simulation results to compare these two with the conventional Parallel Equal (PE) method. We aimed at highly utilisation of I/O bus or I/O channel under the assumptions of being dealt with data bandwidth bounded applications having different input and output data sizes. The proposed algorithms are developed considering some parallel digital video processing applications representing a wide range of applications. These algorithms apply on any data independent algorithm requiring substantial amount of data to process where arbitrary data partitioning is available. In the systems side, an optimal value for the number of processors can be computed for given characteristics of both application and systems which is modeled with

(PI)		PROCESSORS							
		1	2	3	4	5	6	7	8
T_{cycle}		129.0	69.91	51.63	43.56	39.57	37.63	36.85	36.82
PARTITIONS	1	1.00	0.51	0.35	0.28	0.24	0.21	0.20	0.19
	2		0.49	0.33	0.26	0.22	0.19	0.18	0.17
	3			0.31	0.24	0.20	0.18	0.16	0.15
	4				0.22	0.18	0.16	0.14	0.13
	5					0.16	0.14	0.12	0.12
	6						0.12	0.11	0.10
	7							0.09	0.08
	8								0.07
$d_I \geq \ldots$		-	0.05	0.08	0.10	0.13	0.15	0.17	0.20
$d_N \geq \ldots$		-	0.02	0.03	0.04	0.05	0.06	0.07	0.08

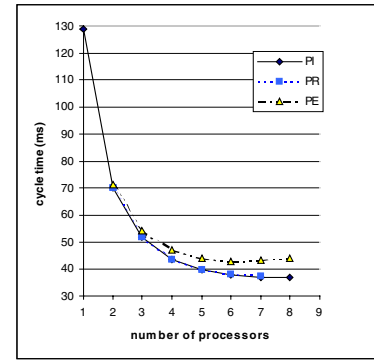

Fig. 4. Data partitions and cycle times for PI and comparison of cycle times

five parameters. Suggested algorithms were evaluated only on a bus based architecture with video based applications in this paper. Hierarchical structures such as tree architectures, mathematical applications such as domain decomposition are yet to be investigated using the same cost model and analysis method.

References

1. Crandall P. E., Quinn M. J., A Partitioning Advisory System for Networked Data-parallel Processing, Concurrency: Practice and Experience, 479-495, August 1995.
2. Agrawal R, Jagadish H V, Partitioning Techniques for Large-Grained Parallelism, IEEE Transactions on Computers, Vol.37, No.12, December,1988.
3. Culler D, Karp R, Patterson D, Sahay A, Schauser K, Santos E, Subramonian R and Eicken T, LogP: Towards a realistic mode of parallel computation, Proceedings of 4th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, Vol.28, May 1993.
4. Lee C., Hamdi M., Parallel Image Processing Applications on a Network of Workstations, Parallel Computing, 21 (1995), 137-160.
5. Moritz C A, Frank M, LoGPC: Modeling Network Contention in Message-Passing Programs, ACM Joint International Conference on Measurement and Modeling of Computer Systems, ACM Sigmetrics/Performance 98, Wisconsin, June 1998.
6. Weissman J.B., Grimshaw A. S., A Framework for Partitioning Parallel Computations in Heterogeneous Environments, Concurrency: Practice and Experience, Vol.7(5),455-478,August 1995.
7. Altilar D T, Paker Y, An Optimal Scheduling Algorithm for Parallel Video Processing, Proceedings of International Conference on Multimedia Computing and Systems'98, Austin Texas USA, 245-258, July 1998.
8. ISO/IEC, MPEG 4 Video Verification Model Ver7.0, N1642, Bristol, April 1997.