Using UPPAAL to analyze an MPEG−2 algorithm ⋆

M.Emilia Cambronero[1], **Anders P. Ravn**[2], **and Valentín Valero**[1]

[1] Escuela Politécnica Superior de Albacete
Universidad de Castilla-La Mancha, 02071 Albacete, SPAIN
[2] Department of Computer Science, Aalborg University, Aalborg, DENMARK
emicp@infoab.uclm.es, apr@cs.aau.dk, valentin@infoab.uclm.es

Abstract The performance of a parallel algorithm for an MPEG−2 encoding is analyzed using timed automata models in the UppAal tool. We have constructed both a sequential model of MPEG-2, and a parallel model of MPEG-2 and then, a comparison of the results obtained for both models is made. We show how a model checking tool for timed automata is used to find exact bounds on the performance. Finally, we outline a correctness proof for the parallelization of the algoritm using an untimed bisiumulation relation.

1 Introduction

We first present a brief description of the MPEG−2 encoder. Specifically, MPEG−2 is a standard for image encoding for streams of related images [4]. It is intended for a wide range of applications, including Video–on–Demand (VoD), High Definition TV (HDTV) and video communications using broadband net works. The MPEG standards were designed to satisfy:

- The need for a high compression, which is achieved by exploiting both spatial and temporal redundancies within an image sequence.
- The need for random access capability, which is obtained by including format pictures (I pictures), which are encoded with no reference to other frames, only exploiting the spatial correlation in the frame.

MPEG digital video coding techniques are statistical in nature. Specifically, the basic statistical property used by MPEG compression is interpixel region correlation. Under the MPEG standards, correlation search is performed by tracking the datawithin 16×16 pixel regions, called macroblocks.

Given two contiguous frames, f_t *and* f_{t+1}, for each macroblock in f_t, the encoder determines the best matching macroblock inf_{t+1} and calculates the motion vector, which captures the macroblock translation information. Thus, the temporal redundancy reduction processor generates a representation for f_{t+1} using the corresponding macroblock from f_t. This representation contains only the motion vector and the prediction error (changes between both frames). This technique is called motion compensated prediction. Furthermore, spatial redundancies are reduced by a DCT (Discrete Cosine Transform). With this coding process some subjective redundancies in the image are removed, on the basis of human visual criteria. The combination of the two techniques described above is the key element of the MPEG encoding process.

⋆ This work has been supported by the MCyT project "Description and Performance of Distributed Systems and Application to Multimedia Systems, Ref. TIC2003-07848-c02-02"

In order to get random access and high compression, the MPEG-2 standard specifies three types of compressed video frames: I pictures, P pictures and B pictures. An I picture (intracoded pictures) is coded without reference to other frames, exploiting only spatial correlation in a frame. A P picture (predictive coded pictures) is coded using motion compensated prediction of a previous I or P picture. Finally, B pictures (bidirectionally–predictive coded pictures) are obtained by motion compensation using past and future reference frames (I or P pictures). A group of consecutive I, P and B pictures constitute a Group of Pictures (GoP). Therefore, a video sequence is a sequence of GoPs.

A block diagram of the MPEG encoder is shown in Figure 1. In order to understand how the MPEG-2 encoder works, we will consider a typical GoP consisting of the frames IBBP. Despite the B pictures appearing before the P picture, the encoding order is IPBB because B pictures require both past and future frames as references.

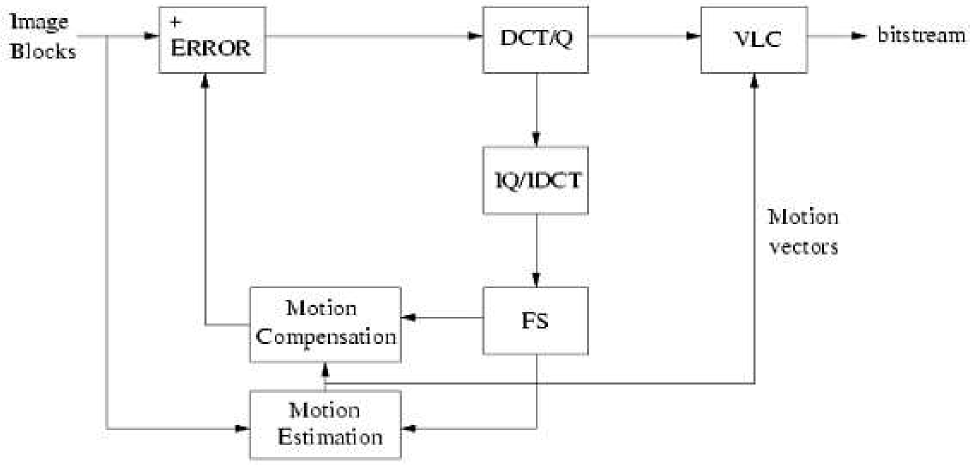

Fig. 1. Block diagram of the MPEG algorithm

The first frame in a GoP (I picture) is encoded in I mode without references to any past or future frames. The DCT is applied to each macroblock and then it is uniformly quantized (Q). After quantization, it is encoded using a variable length code (VLC) and it is sent to the output buffer. At the same time the reconstruction (IQ) of all nonzero DCT coefficients belonging to one macroblock and the Inverse DCT (IDCT) give us a compressed I picture which is stored temporarily in the Frame Store (FS). When the input is coded either as P or B pictures, the encoder does not code the picture macroblocks directly. Instead, it codes the prediction errors and the motion vectors. With P pictures, for each macroblock in the current picture, the motion estimation gives the coordinates of the macroblock in the I picture that best matches its characteristics and thus, the motion vector may be calculated. The motion compensated prediction error is obtained by subtracting from each pixel in a macroblock its motion shifted counterpart in the previous frame. The prediction error and the motion vectors are coded (VLC) and sent to the output buffer. As in the previous case, a compressed P picture is stored in the Frame Store.

With B pictures, the motion estimation process is performed twice: for a past picture (I picture in this case), and for a future picture (P picture).

Prediction errors and both motion vectors for each macroblock are coded (VLC) and sent to the output buffer. Notice that the compressed B pictures are not stored in the Frame Store, since they are not needed to calculate any other pictures.

It is clear from the description that major parts of the encoding of contiguous frames may proceed in parallel, and that some of the internal operations, as VLC/OUT for the I and P frame, and all the process for codifying the second frame B can also proceed in parallel. Thus, a parallel algorithm would improve the performance of the encoding process significantly. However, the performance of the resulting algorithm is hard to analyze. Therefore, we have modelled it using Timed Automata [1, 2], and we have used the model checker of the UPPAAL tool [5, 6] to derive exact performance bounds on the parallel version of the algorithm.

In the next section we present the particular model we have used to describe the encoder, both for the sequential and the parallel version. Then, in Section 3 we show that both models are in fact equivalent. Finally, in Section 4 we conclude and discuss future work.

2 Timed automata for MPEG−2

UPPAAL is a tool box for modeling, simulation and verification of real-time systems. This tool allows us to check the performance of timed automata models of the sequential and parallel algorithms of MPEG-2. This is done using model checking with the verifier of the UPPAAL tool.

2.1 Sequential model

We assume that our GoPs are formed by blocks IPBB. In the sequential model these frames are coded consecutively. The frame I is first coded, next the frame P and then the B frames. ¿From the block diagram of Figure 1, we can obtain the automaton of Figure 2. The algorithm starts at the state marked with a double circle (Read_I). The I−frame is encoded through the states: Read_I, DCT_Q_I, OUT_I. The P−frame is encoded along the states from IQ_DCT_I to OUT_P. Finally, the two B−frames are encoded in states from IQ_DCT to OUT_B1 and from EST_B to OUT_B2.

Timing is introduced through clock variables, which range over real values. In the example of Figure 2 there are two clocks, x and w. Clock x is used to bound the time spent in individual states and w is used to measure the time for encoding a complete block. Clocks are manipulated as follows: For example, the state DCT_Q_I has one ingoing transition in which x is *reset* to zero (in general, a clock can be reset to a constant non-negative integer value); the outgoing transition has a clock *guard*, x == 595, which means that the transition is only enabled when the clock x is equal to 595 (in general, clock guards can be either boolean combinations of comparisons of an integer constant with a clock or a difference between two clocks). Inside a state time progresses linearly for all clocks, and there is a *state invariant*, here x<=595, which gives an upper bound for the time spent in the state (in general, it is a positive boolean combination of upper bounds on clocks).

Note that when a state has no invariant and no clock guard, it may take an arbitrary amount of time. However, states marked with U are *urgent*; they must be left as soon as possible and they take no time.

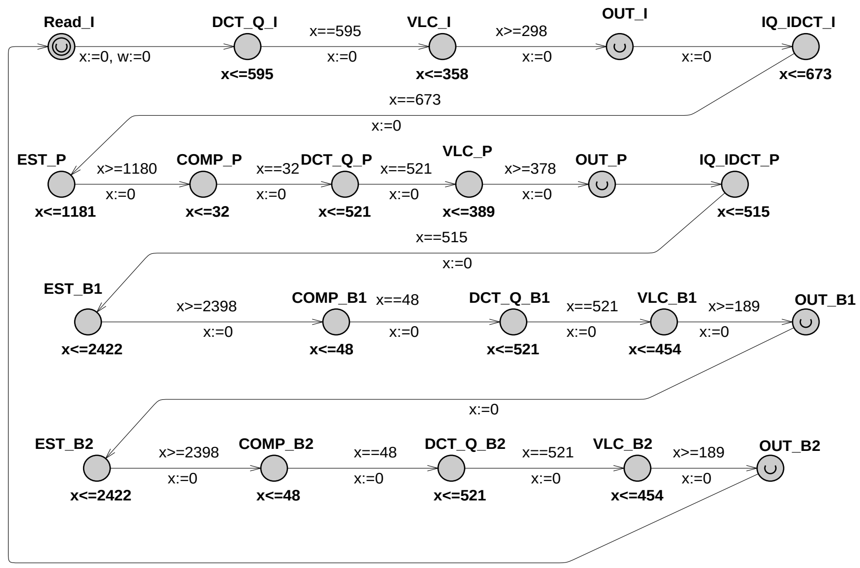

Fig. 2. Model of the MPEG-2 sequential Algorithm

The specified time constants for invariants and guards in Figure2 have been obtained from several real measurements, by coding the so called "Composed" video sequence [3], by a completely software based MPEG-2 video encoder derived from the freely available one developed in Berkeley [9]. The real values thus obtained for I and P pictures reaching the output buffer are shown in Table 1 (measured values), as well as the times for encoding the complete GoP. In order to get values for the different elements of the encoder we have included some hooks in the source code, which correspond to the beginning and the end of the elementary actions that we have described in the specification of the algorithm in Figure 1.

The experiment has been repeated 30 times on a Pentium II processor - 350 MHz and 64 MB RAM, and the values used in Figure 2 are the minimum (upper bound of the invariant) and maximum (upper bound on guard of outgoing transition) of all these trials. In order to avoid interferences in the obtained results, no other programs were running in our experimental setup.

The measured values thus obtained for I and P pictures reaching the output buffer are shown in Table 1, as well as the times for encoding the complete GoP.

Table 1. Times for encoding the pictures

	Measured	
Picture	**Min**	**Max**
I	893ms	953ms
P	3688ms	3738ms
GoP (IPBB)	11567ms	12085ms

[3] The "Composed" video sequence (format PAL CCIR601, 720x576 pixels) is a representative video sequence which has several different motion levels.

We can now use the UPPAAL model checker to get exact bounds for the different frames. For instance, for the complete GoP we use two properties for the global clock w: the first one is an invariant property for the system:

A[] OUT_B2 imply w <= 12085

where A[] is Computational Tree Logic [3] modality meaning for all paths (a path is the finite or infinite sequence of states encountered during a run) and for all states in each path the condition OUT_B2 imply w <= 12085 holds. In other words, whenever the state OUT_B2 is reached, the global clock w is not greater than 12085. The second property is E<>OUT_B2 and w==10504, where E<> means that for some path, there is a state, namely OUT_B2, satisfying OUT_B2 and w==10504.

Once these properties have been verified we know that the column Max in the following Table 2 is definitely an upper bound and the column Min is larger than the lower bound. In order to get exact bounds, we also check A[] OUT_B2 imply w>=10504, i.e. Min is the greatest lower bound and E<>OUT_B2 and w =12085, i.e. that Max is the least upper bound.

Table 2. Times in UPPAAL for encoding the pictures of the sequential algorithm MPEG-2

	Measured	
Picture	**Min**	**Max**
I	893ms	953ms
P	3677ms	3749ms
B1	7348ms	7709ms
B2 (IPBB)	10504ms	11154ms

These results should be compared with those obtained by simulations in Table 1. Already here we see that we improve our bounds, and as we parallelize the algorithm, it becomes more difficult to implement a faithful simulation, and the variation in the measured values are probably greater, thus the model checking approach becomes of real value. In the same vein, should we have reason to doubt the simulation results for the elementary blocks, we could subdivide them and measure more predictable elementary blocks, before modelling the algorithm.

In order to obtain the bounds we have used a binary search technique, starting with an upper bound. The steps that the algorithm follows to obtain the least upper bound are:

- Step 1: check the formula
  F1: A[] at LOC imply w<upper_bound
  for upper_bound = 0, 2, 4,..., 2^m. Until F1 is satisfied, when upper_bound = 2^m
- Step 2: lower_bound := 0
- Step 3: middle := (upper_bound + lower_bound) / 2.
  Check the formula F2:
  A[] at LOC imply w < middle
  If it holds upperbound := middle
  else lower_bound := middle
- STOP: when lower_bound == upper_bound else repeat from Step 3.

The result is the least upper bound.

To calculate the greatest lower bound we use the same algorithm, but in this case we use the formulas:

The upper bound is initialized from the previous algorithm.

2.2 Parallel model

The parallelization exploits some particular characteristics of the MPEG−2 encoding algorithm:

1. Within each block we have parallelized some of the internal operations, as VLC/OUT for the I and P frame, and all the process to encode the second frame B. Then, we need 4 processors for executing each part within every block:
   - Main, in which we have the encoding of frame I (except the part of VLC/OUT), encoding of frame P (except the part of VLC/OUT), and the encoding of the first frame B. See Figure 3.
   - Part VLC/OUT for the frame I. See Figure 4.
   - Part VLC/OUT for the frame P. See Figure 5.
   - Encoding of the second frame B. See Figure 6.
   - A reader process is also added to ensure mutual exclusion around reading a frame.
2. On the other hand, we duplicate the encoding of blocks. In this way, we need 4 processors to execute the 4 blocks IPBB, which are now executed in parallel.

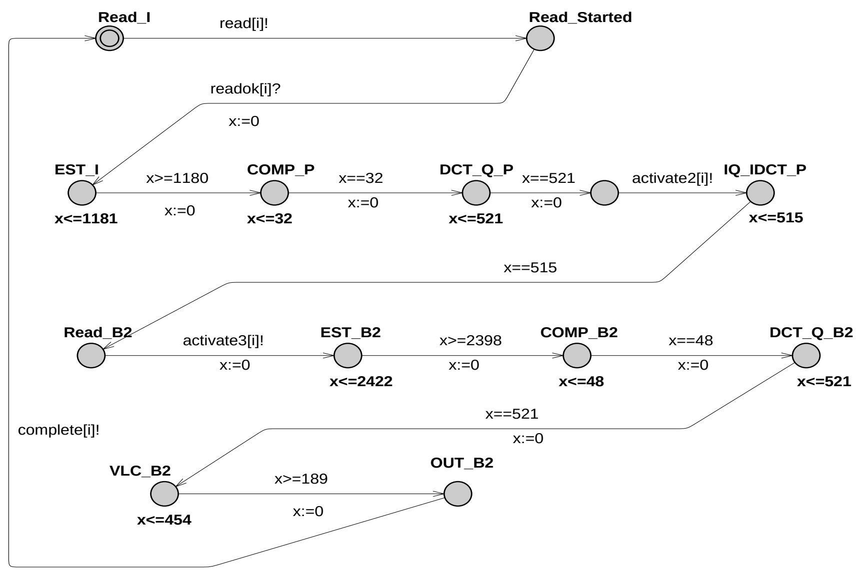

Fig. 3. Model of Main in Parallel Algorithm MPEG-2

We have used channels: read[i], activate1[i], activate2[i], activate3[i], complete[i], to synchronize the paralle processes. Here, “i” is the number of a parallel main loop, to this variable allows us to execute several main loops concurrently.

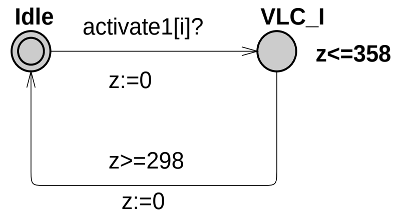

Fig. 4. Model of ObtainI in Parallel Algorithm MPEG-2

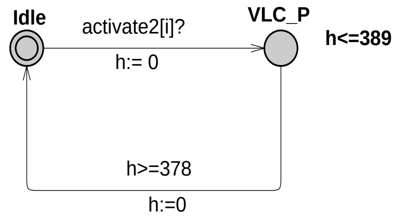

Fig. 5. Model of ObtainP in Parallel Algorithm MPEG-2

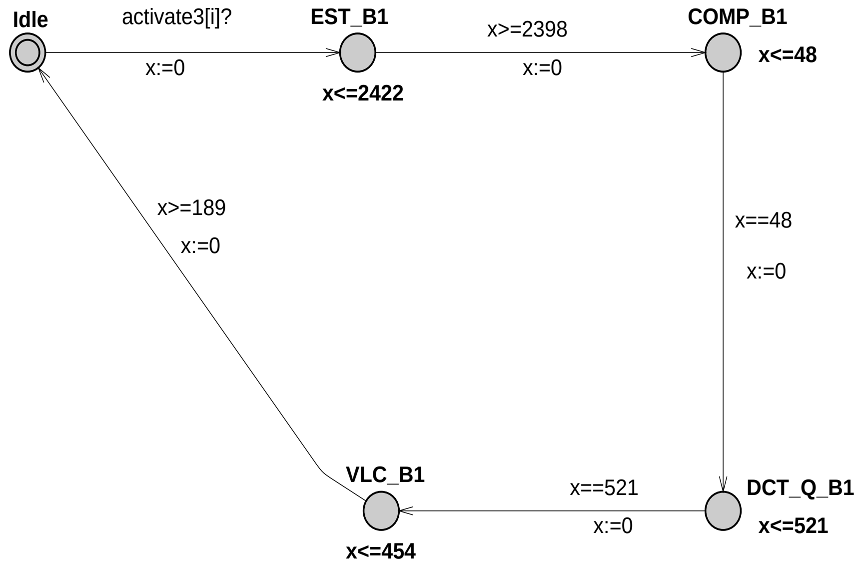

Fig. 6. Model of ObtainB1 in Parallel Algorithm MPEG-2

In the parallelization we need 4 processors. When we introduce more main loops, we need 4*MAX where i ranges from 1 to MAX. The reader process is not a real processor, it models a critical region around reading the next block.

Now, we can use again the model checker of UPPAAL to obtain the corresponding bounds for this parallel version of the encoder. Thus, using the previous described algorithm we have obtained the results reported in Table 3.

Table 3. Times in UPPAAL for encoding the pictures for the parallel version of algorithm MPEG2

IPBB Block	Sequential		Parallel		Gain%		Speedup	
Iteration	Min	Max	Min	Max	Min	Max	Min	Max
1:IPBB	10504	11154	6673	8231	64	78	1.20	1.33

When we compare the results obtained for both versions, we can see the potential improvement that we obtain with the parallel version with respect to the sequential one. More specifically, we know that the time required for encoding the complete GoP is the time required to finish the encoding of both B1 and B2. Thus, the parallel version requires a time in the interval [6673 ms, 8231ms]. With just one iteration, the speedup is between 1.20 and 1.33. This rather low speed up is caused by the sequential execution of part of I and part of P coding.

Table 4. Times in UPPAAL for encoding the pictures for the parallel version of algorithm MPEG2

IPBB Block	Sequential		Parallel		Gain%		Speedup	
Iteration	Min	Max	Min	Max	Min	Max	Min	Max
2:IPBB	21008	22308	7941	13925	38	66	1.43	2.25

With two parallel encoders, see Table 4, the gain obtained when parallelizing algorithm MPEG-2 increases. More experiments with a larger MAX indicate that it settles to a value of 40% for the minimum and 50% for the maximum.

3 Correctness

It is necessary to demonstrate that our systems are equivalent in order to justify the gain obtained with the parallel model with respect to the sequential model. For that purpose we have developed a bisimulation relation between the two systems. Bisimulation equivalence is a semantic equivalence relation on distributed systems. It identifies systems with the same branching structure [11].

Bisimulation equivalence is defined on the states of a given Labelled Transition System (LTS), or equivalently between different process graphs.

Definition 1 A LTS = (S; A; $\rightarrow$) contains S a set of states, and A a set of labels. The transition relation is $\rightarrow \;\subseteq S\times A\times S$. For given states s, s' and label a, (s,a,s') $\in\rightarrow$ is often written $s \stackrel{a}{\longrightarrow} s'$.

One may add more structure by identifiying initial states S0, distinguishing an internal transtion label, distigusihing control states from variables, etc.

Definition 2 Let L and L' be labelled transtion systems. A bisimulation is a binary relation R $\subseteq$ S$\times$S', satisfying:

- if sRs' and $s \xrightarrow{a} t$ with a $\in$ A, then there exists a t' with $s' \xrightarrow{a} t'$ and tRt',
- if sRs' and $s' \rightarrow t'$ with a $\in$ A, then there exists an t with $s \xrightarrow{a} t$ and tRt'.

In the table of Figure 7 it is possible to see how we build up a bisimulation relation that demonstrates the equivalence of the developed models. In a first step, we show that the decoupling of read into a separate reader process is ok.

It is easy to check that the conditions for a bisimulation are satisfied.

For the functional parallelization we take advantage of the local nature of bisimulation. In showing that parallelizing e.g. ObtainI, we can identify all states in Sequential except DCT_Q_I, VLC_I, OUT_I and IQ_IDCT_I which corresponds to active states of ObtainI. The corresponding tables are rather large,and not of general interest, so we omit the further steps.

<table>
<tr><th>Sequential</th><th>Sequential1 || Reader</th><th>a→a'
a R b
b→b' => a'→b'</th></tr>
<tr><td>ReadI</td><td colspan="2">(ReadI, Read)</td></tr>
<tr><td>DCT_Q_I</td><td colspan="2">(Read_Started,DCT_Q_I)</td></tr>
<tr><td>(VLC_I, OUT_I)</td><td colspan="2">(Read_Started, VLC_Started)</td></tr>
<tr><td>IQ_IDCT_I</td><td colspan="2">(Read_Started, IQ_IDCT_I)</td></tr>
<tr><td>EST_P</td><td colspan="2">(Read_Started,FS_I)
(EST_P, Read)</td></tr>
<tr><td>COMP_P</td><td colspan="2">(COMP_P, Read)</td></tr>
<tr><td>DCT_Q_P</td><td colspan="2">(DCT_Q_P, Read)</td></tr>
<tr><td>(VLC_P, OUT_P)</td><td colspan="2">(VLC_Started_P, Read)</td></tr>
<tr><td>IQ_IDCT_P</td><td colspan="2">(IQ_IDCT_P, Read)</td></tr>
<tr><td>EST_B1
COMP_B1
DCT_Q_B1
VLC_B1
OUT_B1</td><td colspan="2">(ReadB2, Read)</td></tr>
<tr><td>EST_B2</td><td colspan="2">(EST_B2, Read)</td></tr>
<tr><td>COMP_B2</td><td colspan="2">(COMP_B2, Read)</td></tr>
<tr><td>DCT_Q_B2</td><td colspan="2">(DCT_Q_B2, Read)</td></tr>
<tr><td>VLC_B2</td><td colspan="2">(VLC_B2, Read)</td></tr>
<tr><td>OUT_B2</td><td colspan="2">(OUT_B2, Read)</td></tr>
</table>

Fig. 7. Bisimulation relation between Sequential and a copy with a separate Reader process

4 Conclusions and Further work

We have shown a general procedure for finding exact execution time bounds for a complex parallel algorithm by means of model checking of a timed automata model. As shown by the MPEG-2 example the procedure is viable for realistic examples and is more precise than even intensive simulation. Checking the properties may require some knowledge of the particular model checker in order to guide it away from potential state explosion problems. The procedure which we have given may be automated in a model checker, something under consideration for UppAal [K.G. Larsen private communication].

Furthermore, we have demonstrated how paralellizations steps may be shown correct by establishing a bisimulation relation between the automata models.

The concrete results for a paralellization of MPEG-2 shows that we can obtain an efficient template for a parallel MPEG-2 algorithm, since we have obtained that from the second iteration the gain obtained is converging to a value of the 40% for the minimum and 50% for the maximum.

Our work in progress is focused to the study of possible paralellization of the more recent versions of the MPEG algorithm, like MPEG-4, which is very used currently, but the implementation is quite different from that one of MPEG-2, so it requires a new paralellization and renewed measurements for basic blocks.

References

1. R. Alur, C. Courcoubetis and D. L. Dill. *Model-Checking for Real-Time Systems*. In Proceedings of the 5th Annual Symposium on Logic in Computer Science, pages 414-425. IEEE Computer Society Press. 1990.
2. R. Alur and D. L. Dill. *Automata for Modelling Real-Time Systems*. In Proceedings of ICALP'90, number 443 in Lecture Notes in Computer Science, pages 322-335. Springer-Verlang. 1990.
3. Edmund M. Clarke, Orna Grumberg and Doron A. Peled. *Model Checking*. MIT Press. 2000.
4. *Draft International Standard Generic Coding of Moving Pictures and Associated Audio. Recommendation H.262* ISO/IEC 138182.
5. K. Havelund, K. G. Larsen, and A. Skou. *Formal Verification of a Power Controller Using the Real-Time Model Checker UPPAAL*. In the Proceedings of the 5th International AMAST Workshop on Real-Time and Probabilistic Systems. 1999.
6. K. Havelund, A. Skou, K. G. Larsen, and K. Lund. *Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL*. 1997.
7. K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. *Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction*. In the Proceedings of the 18th IEEE Real-Time Systems Symposium, pages 14-24. IEEE Computer Society Press. 1997.
8. K. G. Larsen, P. Pettersson, and W. Yi. *UPPAAL in a Nutshell In Springer International Journal of Software Tools for Technology Transfer 1(1+2)*. 1997.
9. MPEG Home Page: http://www.mpeg.org.
10. V. Valero, F.L.Pelayo, F. Cuartero, D.Cazorla *Specification and Analysis of the MPEG2 Video Encoder with Timed-Arc Petri Nets*. In Proc. Int. Workshop on Formal methods for industrial critical systems (FMICS'2002). Electronic Notes in Computer Science, 66.2. 2002.
11. R.J. van Glabbeek. Computer Science Department, Stanford University, CA 94305-9045, USA.
12. W. Yi, P. Pettersson, and M. Daniels. *Automatic Verification of Real-Time Communicating Systems by Constraint Solving*. In Dieter Hogrefe and Stefan Leue, editors, Proceedings of the 7th International Conference on Formal Description Techniques, pages 223-238. North-Holland. 1994.