# PERFORMANCE ANALYSIS OF LOW BIT RATE H.26L VIDEO ENCODER

*Antti Hallapuro[1], Ville Lappalainen[1], and Timo D. Hämäläinen[2]*
[1]Nokia Research Center, P.O. Box 100, FIN-33721, Tampere, Finland
[2]Tampere University of Technology, Digital and Computer Systems Laboratory
E-mail: ville.lappalainen@nokia.com

## ABSTRACT

A new video encoder proposal, H.26L, is compared against H.263 and H.263+. In the comparison, both computational complexity and compression performance are analyzed. Moreover, the trade-off possibilities between the complexity and compression performance within H.26L are presented. Experimental comparisons with H.263 and H.263+ show that H.26L reduces the output bit rate about 30% with the same quality. The computation time increases about three times compared to H.263 and leads into the encoding speed of 3-6 fps for QCIF sequences on a 400 MHz Pentium III processor. Real-time operation can be achieved by applying additional, algorithmic and platform-specific optimizations.

## 1. INTRODUCTION

ITU-T H.26L is a new, emerging video coding standard proposal. It is aimed at very low bit rate, real-time, low end-to-end delay, and mobile applications such as conversational services and Internet video. H.26L is intended to replace the previous H.263 and H.263 Version 2 (H.263+) standards [3] by targeting far beyond in compression performance and advanced features like error resilience. The final major feature adoptions to H.26L are to take place in the beginning of 2001, and the standardization in May 2002 [4]. Currently, the TML-5 (H.26L Test Model Long Term Number 5) document [5] forms the base for the future standard. Additionally, it contains issues relevant for the reference implementation of the encoder.

This paper presents a performance analysis of the emerging standard on a general-purpose processor. A public reference encoder (v. 4.3) is used as a starting point [11], and optimizations in C language are applied. The goal is to compare our H.26L implementation against H.263 and H.263+ and to show how the compression performance improves at the expense of increased complexity. We apply similar optimizations to both H.26L and H.263/H.263+ encoders, to obtain accurate comparisons. The complexity of H.263 and H.263+ encoding has been studied in [9] and [1], for example.

In the following, we summarize H.26L and describe computationally intensive operations in more detail. In sections 3 and 4, we describe the implemented H.26L encoder and present experimental results, respectively. Concluding remarks are given in the final section.

## 2. H.26L ALGORITHM

H.26L is based on a motion compensated hybrid DCT algorithm such as the H.263 and MPEG-4 [2] standards. H.26L uses the traditional translational motion model, although other models such as the affine model have been proposed and studied [7], [10]. However, H.26L provides several enhancements. Firstly, only one regular VLC table is used for symbol coding. Secondly, fractional (¼) pixel precision is used in motion compensated prediction. Thirdly, *seven different block sizes* (16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4) are used for motion compensated prediction. Fourthly, prediction error coding is based on 4x4 blocks and an *integer transform* is used. Finally, multiple reference frames may be used for prediction. Additionally, a deblocking filter is used inside the coding loop. Compared to H.263+, the filter is more complex and it operates on 4x4 block edges instead of 8x8 edges.

Although the implementation of the encoder will not be specified in the final standard, the current TML suggests using the *Hadamard transform* during motion estimation. The use of a transform before computing the measure of prediction error is based on the fact that the prediction error is always transformed during the encoding. Hadamard transform is chosen due to its simplicity. Although allowed by H.26L, we will not consider motion estimation based on multiple reference frames, because it will significantly increase the complexity. Otherwise, we will comply with the reference implementation in this study.

According to the profiling of the reference implementation, *motion estimation* for all the block sizes as well as *image interpolation* resulting in fractional pixel precision consume majority of the encoding time. Motion estimation is highlighted in the following, as it is the most time-consuming operation.

### 2.1. Motion estimation for different block sizes

Compared to H.263, where only two block sizes (16x16 and 8x8) are used, seven different block sizes provide a potentially smaller prediction error at the expense of increased complexity. If this feature is fully utilized, all block sizes are tested and the one giving the best result is chosen. There are two measures of prediction error used in H.26L: SAD (Sum of Absolute Differences) and SATD (T stands for Transformed). They are defined in (1) and (2), respectively.

$$SAD = \sum_{(x,y)\in B_k} \left| I_{wxh}(x,y) - P_{wxh}(x,y) \right| \quad (1)$$

$$SATD = \sum_{(x,y)\in b_k} \left| T_{4x4}(x,y) \right|, \; T_{4x4} = H(D_{4x4}) \quad (2)$$

In (1), $w$ and $h$ denote the width and height of the current frame $I_{wxh}()$ and the predicted frame $P_{wxh}()$, and $B_k$ represents a set of pixels inside a given block. The number of pixels in $B_k$ ranges from 16 to 256, depending on the block size.

In (2), $b_k$ represents a set of pixels inside a given 4x4 block, $D_{4x4}$ denotes the difference block, i.e., $D_{4x4} = \{x, y \in b_k \mid I_{wxh}(x,y) - P_{wxh}(x,y)\}$, and $H()$ denotes the 2-D Hadamard transform. The

1-D Hadamard transform for pixels *a*, *b*, *c*, and, *d* into transform coefficients *A*, *B*, *C*, and *D* is defined in (3). The 2-D transform is obtained by performing the transform both horizontally and vertically. SATD for larger block sizes are obtained by dividing the block into 4x4 subblocks and summing up the SATD values of these subblocks.

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \quad (3)$$

Depending on the block size, a 16x16 macroblock contains 1, 2, 4, 8 or 16 blocks. For each block in a macroblock, three block-matching operations are performed: integer, ½, and ¼ pixel precision block-matching. The integer pixel search is performed inside the search window according to the used block-matching algorithm, while both fractional pixel searches are performed for only nine pixel positions (around the starting position). The integer search starts from the position pointed by the predicted motion vector, which is found according to a set of rules, while the ½ precision search starts at the position where the integer search points. Similarly, the ¼ precision search starts where the ½ precision search points. The search range, i.e., the width of the quadratic search window, is reduced to half from the original one for all block sizes except 16x16, for which the original range is used. For integer search, SAD is used, while for both fractional pixel searches, SATD is used. After all searches are performed, the one resulting in the smallest prediction error is chosen.

# 3. IMPLEMENTATION

Our current implementation is based on the public reference implementation. The major differences are general optimizations such as loop unrolling for SAD calculations and function inlining for SAD and SATD calculations. Also, a look-up table is used for the absolute value computations during SAD calculations.

For comparisons, we used an H.263/H.263+ encoder implementation described in [8]. Both H.26L and H.263 encoders are implemented in C, and they contain neither algorithmic nor platform-specific optimizations. Both use the full-search block-matching algorithm. Although there still exists lots of optimization possibilities, further optimizations were not needed for the purposes of this study.

Experiments were carried out on a 400 MHz Intel Pentium III (Katmai) processor with 128 MB of memory and a 100 MHz system bus. The on-chip L1 data and instruction caches were 16 KB each. The size of the unified, off-chip L2 cache was 512 KB. The encoders were run under Microsoft Windows NT 4.0 (build 1381) with Service Pack 5 (RC 1.1) installed. The encoders were compiled using Microsoft Visual C++ 6.0, with the optimizations targeted at maximizing speed. During the measurements the priority of the encoder process was set to the maximum value and no other applications were allowed to run.

Intel Vtune Performance Analyzer 4.0 was used to get the profile information and to find out the most time-consuming operations.

# 4. RESULTS AND ANALYSES

The profiling of the H.26L reference implementation with the *Foreman* input sequence at 24 kbps gives the following initial results of the distribution the total execution time: 1) Motion estimation including all the operations described in Section 2, 84%, and 2) Image Interpolation, 4%. The remaining 12% is spent on the integer transform and quantization and other minor operations. This gives a clear picture of the critical parts for further optimization.

Our implementation performs roughly twice as fast as the reference encoder, while still achieving similar video quality. However, the main goal in this study is not to make comparisons between our implementation and the reference implementation, for which reason we consider the issue no longer.

In the following, we analyze the performance of our H.26L encoder at both the application and kernel level, and compare its complexity to H.263 and H.263+. Comparison criteria are first discussed.

## 4.1. Comparison criteria

We chose the following approach to compare H.26L and H.263. Firstly, we took H.26L with 7 block sizes as the baseline case, as it represents the best video quality among all the cases. Secondly, the baseline case achieved the desired target bit rate most accurately, while the other cases achieved a bit rate, which usually exceeded the target. Thirdly, we tried to obtain similar luminance (Y) PSNR values for H.26L and H.263(+) encoded sequences. Here, we assume that similar Y PSNR values result in approximately the same spatial video quality. Finally, we fixed the temporal video quality, measured in the frame rate of a sequence. To summarize, we fixed the spatial and temporal video quality, while letting the bit rate and complexity vary.

There are also other ways to perform this kind of comparison such as 1) fixing the bit rate while letting the quality vary, and 2) calculating a combined PSNR instead of separate Y, U, and V values and fixing the combined PSNR while letting the bit rate vary. Our approach has two major advantages compared to the two other approaches. Firstly, our approach clearly shows how H.26L improves the compression ratio over H.263. Secondly, we wanted to analyze the chrominance PSNR values separately to get more detailed information. Moreover, one interesting approach is to fix the complexity of the encoding task. However, we could not follow this approach, because the complexity of H.26L clearly exceeded that of H.263 encoding.

## 4.2. Application-level analysis

Using the H.26L and H.263 encoders, we encoded the following QCIF (luminance resolution: 176x144) sequences: *Akiyo*, *Carphone*, *Foreman*, and *Mother & Daughter*. These sequences were selected from a set of test sequences used during the standardization of H.263 and MPEG-4.

The encoding parameters were as follows: 1) target bit rate was either 8, 14, or 24 kbps, 2) input, i.e., reference frame rate was 30 fps, 3) output, i.e., target frame rate was constant: 10 fps, 4) search window of size 31x31 was used, and 5) number of frames in the encoded sequence was 100, which corresponds to 300 frames in the original sequence, with these reference and output frame rates. Due to the different nature of the sequences, i.e., the amount of motion and spatial details, a unique set of target bit rates was selected for each sequence.

The Advanced Prediction and Unrestricted Motion Vector modes were used for H.263. In addition to these two modes, three additional modes were used for H.263+: Deblocking Filtering, Advanced Intra Coding, and Modified Quantization modes. This selection of modes represents the best video quality available in H.263 and H.263+.

For H.26L, we used the following combinations of block sizes during the motion estimation: 1) 16x16, 2) 16x16 and 16x8, 3) 16x16, 16x8, and 8x16, 4) 16x16, 16x8, 8x16, and 8x8, 5) 16x16, 16x8, 8x16, 8x8, and 8x4, 6) 16x16, 16x8, 8x16, 8x8, 8x4, and 4x8, as well as 7) 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4. Only one reference frame was used in H.26L.

Both encoders encoded exactly the same frames. A fixed quantization parameter ($QP_I$) for the first frame as well as for all subsequent frames ($QP_P$) was used. For each experimentation, we used the pair ($QP_P$, $QP_I$) which resulted in the bit rate closest to the target. The reason for not using a more sophisticated rate control mechanism was to obtain the most accurate performance comparisons. A sophisticated rate control mechanism is able to use variable output frame rate and variable QP for each macroblock, for example.

The variability of the experimental results was taken into account by performing each experiment three times in a row and averaging the results. If the standard deviation exceeded a threshold value, new experiments were performed.

The encoding speed was measured by using the ANSI C clock function. The harmonic mean of the encoding speed was used instead of the arithmetic mean used in the kernel-level analysis, because the arithmetic mean can be justified for the reciprocal of the encoding speed [6].

Table 1 shows the individual encoding speeds (harmonic mean) of all nine experiments for each encoder. For H.26L, only two combinations of block sizes are shown, to represent the most complex and the least complex case. Moreover, this Table shows the average encoding speeds over all experiments.

**Table 1.** Individual and average encoding speeds (frames/s).

| Sequence@ bit rate (kbps) | H263+ | H263 | H26L 1 block | H26L 7 blocks |
|---|---|---|---|---|
| Akiyo @ 8 | 7.1 | 7.3 | 6.1 | 2.6 |
| Akiyo @ 14 | 9.4 | 9.7 | 6.4 | 2.7 |
| Akiyo @ 24 | 10.1 | 10.0 | 6.3 | 2.6 |
| Carphone @ 14 | 5.3 | 5.4 | 5.9 | 2.6 |
| Carphone @ 24 | 5.8 | 5.9 | 5.8 | 2.4 |
| Foreman @ 24 | 5.2 | 5.3 | 5.3 | 2.2 |
| M. & D. @ 8 | 6.0 | 6.4 | 6.3 | 2.9 |
| M. & D. @ 14 | 6.8 | 7.0 | 6.1 | 2.7 |
| M. & D. @ 24 | 7.3 | 7.5 | 6.1 | 2.5 |
| Avg. speed (frames/s) | 6.7 | 6.8 | 6.0 | 2.6 |

H.26L encoding with 1 and 7 block sizes is 1.2X and 2.8X more complex or slower than H.263 encoding, respectively. When comparing to H.263+, H.26L encoding with 1 and 7 block sizes is 1.2X and 2.7X more complex.

Table 2 shows the obtained bit rates for H.263+, H.263 and H.26L. In addition, the Table shows the proportional reduction in bit rate for each case, compared to H.263+.

On the average, H.26L encoding with 1 block size reduces the bit rate by 20% and with 7 block sizes 28% (ranging from 17% to 47%). The reduction in bit rate varies heavily, and it depends on both the used QP and the nature of the sequence. For the same Y PSNR values, H.263 achieves slightly smaller bit rate than H.263+, because H.263+ achieves considerably higher chrominance PSNR values, as shown below in Table 4.

The differences in video quality, in terms of PSNR, between the baseline and the other cases as well as the PSNR values for the baseline case are reported in Table 3 and Table 4.

Tables 3 and 4 prove that we were able to obtain similar Y PSNR values for all cases, as the standard deviation ranges from 0.1 to 0.2 dB. At the same time, it was not possible to obtain similar chrominance values. Thus, U and V PSNR values for H.263 and H.263+ vary more, as can be seen.

**Table 2**. Obtained bit rates (kbps) and bit rate reduction (%).

| Sequence@ bit rate (kbps) | H263+ Bit rate | H263 Bit rate | H26L 1 block Bit rate | H26L 7 blocks Bit rate | H26L 7blocks Reduc-tion (%) |
|---|---|---|---|---|---|
| Akiyo @ 8 | 11.6 | 11.0 | 9.4 | 8.1 | 30.6 |
| Akiyo @ 14 | 27.0 | 27.1 | 17.1 | 14.2 | 47.4 |
| Akiyo @ 24 | 44.3 | 42.1 | 29.7 | 24.4 | 45.0 |
| Carphone@ 14 | 17.8 | 16.1 | 14.6 | 13.9 | 21.7 |
| Carphone@ 24 | 29.2 | 27.5 | 26.5 | 24.3 | 16.8 |
| Foreman @ 24 | 29.9 | 28.2 | 26.8 | 24.7 | 17.5 |
| M. & D. @ 8 | 9.8 | 9.5 | 8.6 | 8.1 | 17.0 |
| M. & D. @ 14 | 18.6 | 18.0 | 15.8 | 14.1 | 23.9 |
| M. & D. @ 24 | 37.1 | 35.6 | 27.7 | 24.3 | 34.7 |
| Avg. reduction | 0.0% | 4.6% | 19.8% | 28.3% | 28.3% |

**Table 3**. Y, U, and V PSNR values (dB) for H.26L baseline case and PSNR differences (dB) for another H.26L case.

| Sequence@ | H.26L 7 blocks | | | H. 26L 1 block | | |
|---|---|---|---|---|---|---|
| bit rate (kbps) | Y | U | V | Y | U | V |
| Akiyo @ 8 | 34.0 | 37.9 | 39.4 | -0.2 | 0.0 | 0.0 |
| Akiyo @ 14 | 38.2 | 42.2 | 42.9 | -0.3 | 0.0 | -0.1 |
| Akiyo @ 24 | 40.9 | 44.3 | 44.9 | -0.2 | -0.1 | -0.1 |
| Carphone@ 14 | 28.0 | 36.2 | 36.6 | -0.2 | 0.0 | -0.1 |
| Carphone@ 24 | 30.7 | 37.3 | 37.8 | -0.2 | 0.0 | 0.0 |
| Foreman @ 24 | 28.6 | 37.6 | 37.8 | -0.2 | -0.2 | -0.1 |
| M. & D. @ 8 | 32.0 | 39.8 | 40.5 | -0.1 | 0.0 | 0.1 |
| M. & D. @ 14 | 34.7 | 41.0 | 41.6 | -0.2 | -0.1 | -0.1 |
| M. & D. @ 24 | 37.2 | 42.7 | 43.2 | -0.2 | -0.1 | -0.1 |
| Avg. diff. (dB) | 0.0 | 0.0 | 0.0 | -0.2 | -0.1 | 0.0 |
| St. dev. (dB) | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.1 |

**Table 4**. PSNR differences (dB) for H.263 and H.263+.

| Sequence@ | H.263 | | | H.263+ | | |
|---|---|---|---|---|---|---|
| bit rate (kbps) | Y | U | V | Y | U | V |
| Akiyo @ 8 | 0.0 | -1.1 | -0.3 | -0.2 | 0.0 | 0.5 |
| Akiyo @ 14 | 0.2 | 0.8 | 0.6 | -0.1 | 0.8 | 0.8 |
| Akiyo @ 24 | -0.3 | -0.1 | -0.3 | -0.2 | 0.6 | 0.4 |
| Carphone@ 14 | -0.1 | -2.3 | -1.8 | 0.0 | -0.1 | -0.1 |
| Carphone@ 24 | -0.1 | -1.0 | -1.1 | -0.1 | 0.0 | -0.3 |
| Foreman @ 24 | 0.3 | -1.9 | -2.4 | 0.1 | -0.4 | -0.8 |
| M. & D. @ 8 | 0.0 | -2.1 | -2.0 | 0.0 | -1.0 | -0.8 |
| M. & D. @ 14 | 0.0 | -0.8 | -1.0 | -0.1 | -0.1 | -0.3 |
| M. & D. @ 24 | -0.1 | -0.9 | -0.9 | -0.1 | -0.4 | -0.4 |
| Avg. diff. (dB) | 0.0 | -1.1 | -1.0 | -0.1 | -0.1 | -0.1 |
| St. dev. (dB) | 0.2 | 1.0 | 0.9 | 0.1 | 0.5 | 0.6 |

Table 5 shows the speedup for H.26L encoding with 1 to 6 block sizes, compared to the baseline case of 7 blocks. The average speedup for each case is given as well.

Table 6 shows the proportional increases in bit rate, when using less than 7 block sizes in H.26L encoding, compared to

the baseline case. Also, the table shows the average values for each combination of block sizes.

Tables 5 and 6 clearly show the trade-off possibilities between the encoding speed and compression performance. However, it should be noted that the speedups would be significantly lower, if faster block-matching algorithms were used, or if the motion estimation was further optimized.

**Table 5**. Speedups in encoding speed when using less than 7 block sizes in H.26L encoding.

| Sequence@ bit rate (kbps) | 6 blocks | 5 blocks | 4 blocks | 3 blocks | 2 blocks | 1 block |
|---|---|---|---|---|---|---|
| Akiyo @ 8 | 1.06 | 1.19 | 1.32 | 1.52 | 1.82 | 2.31 |
| Akiyo @ 14 | 1.08 | 1.20 | 1.34 | 1.53 | 1.87 | 2.34 |
| Akiyo @ 24 | 1.10 | 1.22 | 1.39 | 1.62 | 1.96 | 2.46 |
| Carphone@ 14 | 1.06 | 1.17 | 1.30 | 1.49 | 1.81 | 2.25 |
| Carphone@ 24 | 1.08 | 1.21 | 1.36 | 1.59 | 1.94 | 2.45 |
| Foreman @ 24 | 1.08 | 1.23 | 1.38 | 1.62 | 1.97 | 2.46 |
| M. & D. @ 8 | 1.06 | 1.16 | 1.26 | 1.44 | 1.73 | 2.13 |
| M. & D. @ 14 | 1.09 | 1.21 | 1.35 | 1.55 | 1.85 | 2.28 |
| M. & D. @ 24 | 1.10 | 1.22 | 1.39 | 1.62 | 1.96 | 2.44 |
| Avg. speedup | 1.08 | 1.20 | 1.34 | 1.55 | 1.88 | 2.35 |

**Table 6**. Proportional increase (%) in bit rate when using less than 7 block sizes in H.26L encoding.

| Sequence@ bit rate (kbps) | 6 blocks | 5 blocks | 4 blocks | 3 blocks | 2 blocks | 1 block |
|---|---|---|---|---|---|---|
| Akiyo @ 8 | 0.1 | 0.6 | 1.1 | 2.2 | 9.8 | 16.8 |
| Akiyo @ 14 | 0.5 | 1.3 | 3.2 | 5.4 | 12.6 | 20.6 |
| Akiyo @ 24 | -0.2 | 1.0 | 3.1 | 6.7 | 14.7 | 21.7 |
| Carphone@ 14 | -0.1 | 0.9 | 1.1 | 0.6 | 2.7 | 4.9 |
| Carphone@ 24 | 0.0 | 1.1 | 0.9 | 1.0 | 4.0 | 8.8 |
| Foreman @ 24 | 0.1 | -0.2 | 0.1 | 1.1 | 3.0 | 8.8 |
| M. & D. @ 8 | 0.0 | -0.5 | -0.7 | -0.1 | 1.1 | 5.5 |
| M. & D. @ 14 | 0.0 | 1.7 | 1.1 | 1.5 | 4.8 | 12.1 |
| M. & D. @ 24 | 0.7 | 1.5 | 1.3 | 2.1 | 7.4 | 14.2 |
| Avg. incr. (%) | 0.1 | 0.8 | 1.3 | 2.3 | 6.7 | 12.6 |

### 4.3. Kernel-level analysis of motion estimation

The results in this section are for H.26L encoding with 7 block sizes. The encoded sequence is *Foreman* (in QCIF) at 24 kbps. Firstly, the integer pixel block-matching consumes 75%, while the ½ and ¼ pixel precision block-matching consume 11% and 14% of the total execution time spent on motion estimation. These figures are averaged over all 7 block sizes. Secondly, the total time spent on motion estimation is distributed among the different block sizes as follows; 16x16: 32%, 16x8: 12%, 8x16: 13%, 8x8: 12%, 8x4: 10%, 4x8: 11%, and 4x4: 9%. These figures contain both integer and fractional pixel precision block-matching.

Finally, at the lowest level, there are SAD or SATD calculation kernels for each block size. Table 7 shows the average execution time (cycles) for the SAD 16x16 kernel together with the ratio of the average execution time of each kernel compared to the SAD 16x16 kernel.

The execution time of each optimized routine was measured using a special instruction (Read from Time Stamp Counter), which returns clock cycles. Before computing the ratios, the time for each kernel is scaled to cover the whole macroblock area, i.e., the measured time is multiplied by either 2, 4, 8, or 16, depending on how many blocks fit into a macroblock.

Based on Table 7, we see that using SATD is from 9.9X (8x16) to 14.1X (16x16) slower than using SAD, and 11.4X slower on the average. The use of different block sizes for the SAD calculation is from 1.2X to 1.4X slower than the use of the basic macroblock size.

**Table 7**. Ratio of average execution time of each kernel compared to SAD 16x16 kernel.

| Routine | Ratio of execution times | Routine | Ratio of execution times |
|---|---|---|---|
| SAD16x16 | (611 cycles) 1.0 | SATD16x16 | 14.1 |
| SAD16x8 | 1.3 | SATD16x8 | 13.8 |
| SAD8x16 | 1.4 | SATD8x16 | 13.9 |
| SAD8x8 | 1.3 | SATD8x8 | 14.1 |
| SAD8x4 | 1.2 | SATD8x4 | 14.5 |
| SAD4x8 | 1.4 | SATD4x8 | 15.2 |
| SAD4x4 | 1.4 | SATD4x4 | 16.0 |

## 5. CONCLUSIONS

Experimental results show that H.26L can achieve a bit rate reduction of about 30% with the same quality as in H.263(+). However, the encoding speed is about three times lower due to more complex operations. The presented implementation on a 400 MHz Pentium III used neither media ISA (Instruction Set Architecture) extensions nor algorithmic optimizations. Further optimizations will achieve and exceed the real-time encoding speed of 10 fps. Especially the SAD calculation for multiple block sizes should be carefully implemented. In the future, we will compare highly optimized versions of H.26L and H.263(+) to find out and analyze the differences at a very detailed level.

## 6. REFERENCES


[1] G. Côté et al., "H.263+: Video Coding at Low Bit Rates," IEEE Trans. Circuits and Systems for Video Technology, vol. 8, no. 7, pp. 849-866, Nov. 1998.

[2] ISO, MPEG-4 Video Verification Model version 17.0, ISO/IEC JTC1/SC29/WG11 N3515, Beijing, July 2000.

[3] ITU-T, Recommendation H.263, "Video Coding for Low Bit Rate Communication," Feb. 1998.

[4] ITU-T Q.15/SG16, "Meeting report of the eleventh meeting (meeting K) of the ITU-T Q.15/16 Advanced Video Coding Experts Group," Aug. 2000.

[5] ITU-T Q.15/SG16, "H.26L Test Model Long Term Number 5 (TML-5) draft0, " Doc. Q15-K-59, Sept. 2000.

[6] R. Jain, "The Art of Computer Systems Performance Analysis," John Wiley & Sons, Inc., pp.188-189, 1991.

[7] M. Karczewicz et al., "MVC: Advanced low bit rate codec for mobile multimedia," Proc. EUSIPCO-2000, Sept. 2000.

[8] V. Lappalainen, "Implementation of H.263 Video Encoder Using Intel MMX Technology," M. Sc. Thesis, Tampere University of Technology, Aug. 1997.

[9] V. Lappalainen, "Performance Analysis of Intel MMX Technology for an H.263 Video Encoder," Proc. ACM MULTIMEDIA '98, pp. 309-314, Sept. 1998.

[10] V. Lappalainen, "Performance of an Advanced Video Codec on a General-Purpose Processor with Media ISA Extensions," IEEE Trans. Consumer Electronics, vol. 46, no. 3, pp. 706-716, Aug. 2000.

[11] Telenor Broadband Services, "TML H.26L Codec," via ftp://standard.pictel.com/video-site/h26L/, 31st Aug. 2000.