Optimization of H.263 Video Encoding Using a Single Processor Computer: Performance Tradeoffs and Benchmarking

Shahriar M. Akramullah, *Member, IEEE*, Ishfaq Ahmad, and Ming L. Liou, *Fellow, IEEE*

***Abstract*—In this paper, we present the optimization and performance evaluation of a software-based H.263 video encoder. The objective is to maximize the encoding rate without losing the picture quality on an ordinary single-processor computer such as a PC or a workstation. This requires optimizations at all design and implementation phases, including algorithmic enhancements, efficient implementations of all encoding modules, and taking advantage of certain architectural features of the machine. We design efficient algorithms for DCT and fast motion estimation, and exploit various techniques to speed up the processing, including a number of compiler optimizations and removal of redundant operations. For exploiting the architectural features of the machine, we make use of low-level machine primitives such as Sun UltraSPARC's *visual instruction set* and Intel's *multimedia extension*, which accelerate the computation in a *Single Instruction Stream Multiple Data Stream* fashion. Extensive benchmarking is carried out on three platforms: a 167-MHz Sun UltraSPARC-1 workstation, a 233-MHz Pentium II PC, and a 600-MHz Pentium III PC. We examine the effect of each type of optimization for every coding mode of H.263, highlighting the tradeoffs between quality and complexity. The results also allow us to make an interesting comparison between the workstation and the PCs. The encoder yields 45.68 frames per second (frames/s) on the Pentium III PC, 18.13 frames/s on the Pentium II PC, and 12.17 frames/s on the workstation for QCIF resolution video with high perceptual quality at reasonable bit rates, which are sufficient for most of the general switched telephone networks based video telephony applications. The paper concludes by suggesting optimum coding options.**

***Index Terms*—H.263, low-level parallelism, MPEG, real-time video coding, very low bit rate.**

I. INTRODUCTION

H.263 is an international standard optimized for compressing video at low bit rates. It supports efficient transmission of digital video over narrow-band telecommunication channels. Several research results reporting implementation [1], [20] and improvement [4], [12], [15], [16], [21] of the H.263 video codec have been reported. The complexity of the H.263 video coding makes it seemingly impossible to accomplish real-time video coding without using special-purpose hardware, such as function-specific multimedia processors, parallel digital signal processing (DSP) system, programmable single-component mixed-media coprocessors etc., [3]. However, a dedicated hardware implementation is not flexible to incorporate new algorithms and can become obsolete. Real-time performance using software, on the other hand, has only been achieved on a multiprocessor system [1]. In this work, our aim is to build a software-only real-time H.263 encoder on a general-purpose single-processor ordinary computer such as a PC or a workstation.

Optimization of the codec means optimizations at all of the implementation phases, including algorithmic enhancements, compiler and code optimization, and taking advantage of certain architectural features of the machine. We optimize efficient algorithms for various functional modules of H.263 video encoder such as motion estimation (ME), discrete cosine transform (DCT), and inverse DCT, and elevate their performance through efficient implementation. We use compiler optimizations in order to exploit sophisticated scheduling algorithms for redistributing the tasks for fast processing. Performance of the implementation is enhanced by using simplified model of floating-point arithmetic, loop parallelization, common subexpression elimination, copy propagation, automatic register allocation, tracing of the effects of pointer assignments, etc. Our code optimization includes loop unrolling, which decreases the number of iterations, and data type optimization (DTO), which chooses suitable data types of variables in the program's critical path so as to yield the most efficient performance of basic arithmetic operations. In addition, we remove all possible redundant operations.

In order to exploit the architectural features of the machine, we exploit low-level machine primitives that provide extensions to the core instruction sets with a view to support multimedia data. The use of extended instruction sets in existing microprocessors explore potential low-level parallelism in order to enhance performance of applications with low-precision data. Video coding deals with the data streams which are regular and have independent control flow. Thus, data-level parallelism can be explored by introducing additional logic to partition a higher precision data path to handle multiple pieces of packed lower precision data processed with a single instruction. Typical examples of such multimedia-capable general-purpose processors include Intel's Multi Media eXtension (MMX) [17] and Sun Microsystems' Visual Instruction Set (VIS) [22]. Using these ex-

Manuscript received October 20, 1998; revised April 27, 2001. This work was supported by the Hong Kong Telecom Institute of Information Technology and the Hong Kong RGC under Grant HKUST6228/99E. This paper was recommended by Associate Editor S. Panchanathan.

S. M. Akramullah and M. L. Liou are with the Department of Electrical and Electronic Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: eetipu@ee.ust.hk; eeliou@ee.ust.hk).

I. Ahmad is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong (e-mail: iahmad@cs.ust.hk).

Publisher Item Identifier S 1051-8215(01)06745-3.

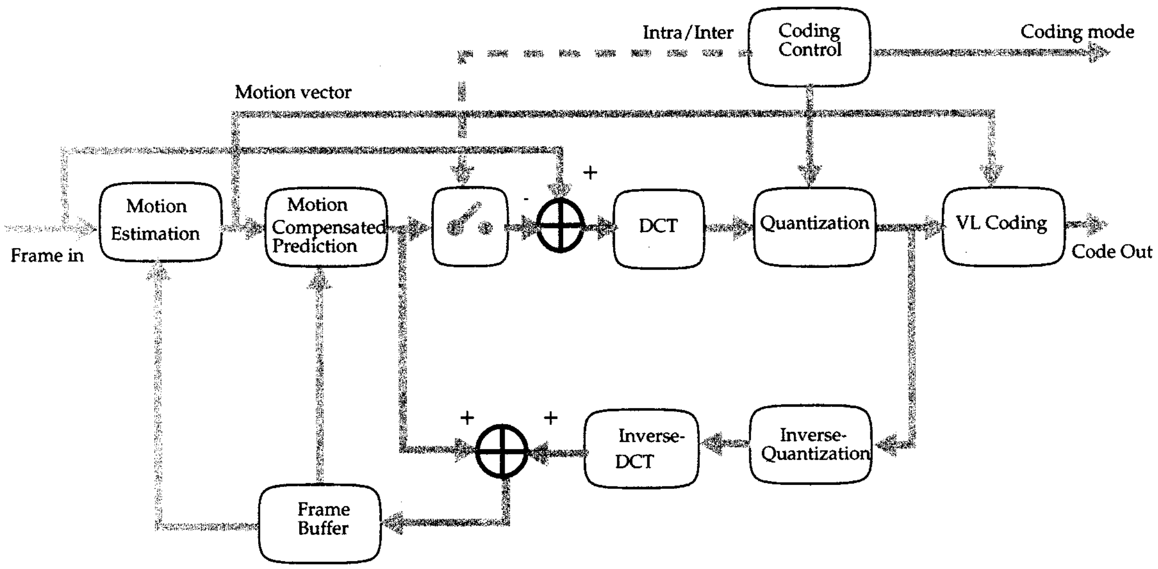


Fig. 1. H.263 computational modules.

tended multimedia instruction sets, we accelerate the computation in a *single instruction stream multiple data stream* (SIMD) fashion, increase the utilization of available registers in the processor, and remove register contentions between data and control variables.

Extensive benchmarking is carried out on a 167-MHz Sun UltraSPARC-1 workstation, a 233-MHz Pentium II PC, and a 600-MHz Pentium III PC to study the performance of the encoder. Based on the benchmarking results, suggestions are made to decide the optimum coding options. We carry out a thorough benchmarking which considers various aspects of our implementation. The study determines the effect of each type of optimization for each coding mode of H.263. Our results indicate the tradeoffs between quality and complexity, as well as make an interesting comparison between the workstation and the PCs. The encoder achieves frame-encoding speeds up to 45.68 frames/s on the PCs and 12.17 frames/s on the workstation for Quarter Common Intermediate Format (QCIF) resolution of video with high perceptual quality at reasonable bit rates, which is sufficient for most of the general switched telephone networks (GSTN)-based video-telephony applications. As the speed of PCs will further increase, video coding will become an integral part of its resources and a useful commodity.

The rest of the paper is organized as follows. Section II gives an overview of the H.263 video coding standard, Section III describes the extended instruction sets for multimedia enhancements, and Section IV gives a discussion of various optimizations. Experimental results are presented in Section V.

II. Overview of the H.263 Video-Coding Standard

H.263 [10], defined by ITU-T, is aimed at low-bit-rate video coding, with the objective to provide significantly better picture quality than its predecessor H.261 [9]. Conceptually, H.263 is network independent and can be used for a wide range of applications, but its target applications are visual telephony and multimedia on low bit-rate networks like the GSTN, integrated services digital network (ISDN), and wireless networks. Some of the important considerations of H.263 are small overhead, low complexity resulting in low cost, interoperability with other existing video-communication standards (e.g., H.261, H.320), robustness to channel errors, quality-of-service parameters, etc. Based on these considerations, an efficient coding scheme has been developed which gives flexibility to manufacturers to make a tradeoff between picture quality and complexity.

The generalized H.263 source encoder is shown in Fig. 1. H.263 uses a hybrid of interpicture prediction to utilize temporal redundancy and transform coding of the residual prediction error signal to reduce spatial redundancy. Although H.263 is closely related to the H.261, it provides the same subjective image quality at less than half the bit rate [5].

The transform coding is done by discrete cosine transform (DCT). The transformed signal is then quantized with a scalar quantizer, and the resulting symbols are variable-length coded and transmitted. At the decoder, the received signal is inverse quantized, and subsequently, inverse transformed to reconstruct the prediction error signal, which is added to the prediction, thus creating the reconstructed picture. The reconstructed picture is stored in a frame buffer and can serve as the reference picture for the prediction of the next picture. The encoder consists of an embedded decoder, where the same decoding operation is performed so that both the encoder and the decoder have the same reconstructed picture.

A picture is divided into macroblocks, since such division results in more efficient coding. Each macroblock consists of four luminance blocks and two spatially aligned color difference blocks. Each of these blocks are of size 8×8 pixels. One or more macroblock rows are combined into a group of blocks (GOB) to enable quick resynchronization after transmission errors. The GOB structure is simpler than that adopted in H.261. The optional GOB headers may or may not be used, depending on the tradeoff between error resilience and coding efficiency [19].

For improved interpicture prediction, the H.263 decoder has the block-motion compensation capability, while its use in the encoder is optional. Using block-motion compensation, interpicture prediction can be improved when the prediction blocks can be taken from different positions in the previous picture. One motion vector is transmitted per macroblock so that the simple translational motion can be compensated for. Half-pixel precision is used for motion compensation, as opposed to H.261, where full-pixel precision and a loop filter are used. Therefore, the visual quality is better compared to H.261 [5]. The motion-vector symbols are transmitted to the decoder after variable-length coding. The bit rate of the coded video may be controlled by preprocessing or by varying the following encoder parameters: quantizer scale size, mode selections, and picture rate.

Further to the core coding algorithm described above, H.263 includes four negotiable coding options: 1) unrestricted motion vectors (UMVs); 2) advanced prediction; 3) PB-frames; and 4) syntax-based arithmetic coding (SAC). The first three options are used to improve interpicture prediction. The fourth is related to lossless coding of the symbols to be transmitted, which may be used instead of Huffman coding. These coding options increase the complexity of the encoder, but improve the picture quality, thereby allowing a tradeoff between picture quality and complexity [19].

The source coder can operate on one of the five standardized picture formats: 1) sub-QCIF (128×96); 2) QCIF (176×144); 3) CIF (352×288); 4) 4CIF (704×576); and 5) 16CIF (1408×1152), covering a large range of spatial resolutions. Support for both sub-QCIF and QCIF formats in the decoder is mandatory, while either one of these formats must be supported by the encoder. This requirement is a compromise between high resolution and low cost.

UMV Mode: In this mode, motion vectors are allowed to point outside the coded picture area. This allows for a better prediction when a small part of the predicted macroblock is located outside the picture area and, therefore, is not available. In case of the prediction of these unavailable pixels, the edge pixels are used instead. With this mode, a gain in quality is achieved, especially for the smaller picture formats if there is motion at near the picture boundaries. Note that, for sub-QCIF picture format, about 50% of all the macroblocks are located at or near the boundary.

Advanced Prediction Mode: In this optional mode, the overlapped block motion compensation (OBMC) is used for the luminance component which reduces the blocking artifacts and thereby improves the subjective video quality. For some of the macroblocks, four 8×8 vectors are used instead of one 16×16 vector, providing better prediction but at the expense of more bits.

PB-Frames Mode: The principal purpose of the PB-frames mode is to increase the frame rate without significantly increasing the bit rate. A PB-frame consists of two pictures coded as one unit. The P-picture is predicted from the last decoded P-picture and the B-picture is predicted both from the last and current P-pictures. Although the names P-picture and B-picture are adopted from MPEG [8], B-pictures in H.263 serve an entirely different purpose. The quality for the B-frames is intentionally kept lower, so as to minimize the overhead of bidirectional prediction, which is important in low bit-rate applications. B-pictures use only 15%–20% of the allocated bit rate, but result in better subjective impression of smooth motion.

SAC Mode: Since H.263 is optimized for very low bit rates, it uses the optional SAC mode, which replaces the variable length coding/decoding (VLC/VLD) using Huffman tables by arithmetic coding/decoding operations in order to reduce the number of bits to be transmitted. While in the normal VLC/VLD process (using Huffman coding), only a fixed integral number of bits must be used for each coded symbol, arithmetic coding removes this restriction, resulting in a reduced bit rate, while at the same time not losing the advantages offered by normal VLC/VLD.

III. Extended Instruction Sets for Multimedia Enhancement

In this section, we discuss two low-level machine primitives, namely the VIS for SUN UltraSPARC workstations and the MMX for Intel Pentium-based PCs. These are, in effect, extensions to the core instruction sets, specifically designed to embody special instructions suitable for multimedia applications. These instruction sets support integer data processing in single instruction stream multiple data stream (SIMD) fashion by utilizing a packed fixed-point integer, where multiple integer words are grouped into a single 64-bit quantity. These 64-bit quantities can be moved into the 64-bit (integer or floating point) registers and processed with a single instruction, providing data parallelism and thereby enhancing the performance.

A. The VIS

The VIS [22] in the Sun UltraSPARC processor is a RISC-like extension to the SPARC V9 instruction set, which provides core instructions that greatly enhance the graphics and image processing capabilities of SPARC processors [11]. We exploit the data alignment and packing capabilities of VIS, along with its various representations of data (for instance, one 64-bit data may represent eight 8-bit partitioned, four 16-bit partitioned, or two 32-bit partitioned data). Thereby, we use a 64-bit register to perform a set of eight 8-bit, four 16-bit, or two 32-bit integer arithmetic in parallel, providing 8-fold, 4-fold or 2-fold speedup, respectively.

In image- and video-processing applications, many computations can be accelerated in a SIMD fashion. The addition of VIS is equivalent to adding a SIMD fixed-point processor [14]. We process four pixels (each represented by 16 bits) using only one VIS instruction, performing either multiplication, addition, subtraction, or logical evaluation. A few special VIS instructions, in addition to the regular instructions, enable the video coding application to speed up by a factor of four or higher [25]. We perform most operations using VIS in floating-point register file, so that substantial register space is available and there are no register contentions between data and control variables. In our experiments, we have used VIS for ME, MCP, DCT, and IDCT.

B. The MMX Instruction Set

Like VIS, Intel's MMX is an extended multimedia instruction set. MMX implements a new high-performance architectural technique and includes new instructions and data types to

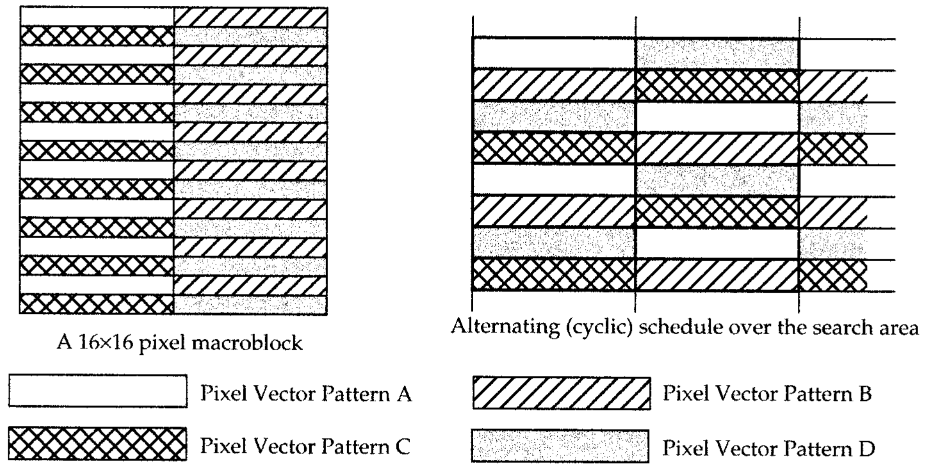


Fig. 2. Pixel-vector subsampling with an alternating schedule of patterns.

achieve increased level of performance on the host CPU. Essentially, MMX exploits the parallelism inherent in many of the algorithms in video, graphics, or multimedia applications by processing several pieces of data with each instruction [17]. MMX introduces 57 new instructions and 8 new virtual 64-bits registers in order to accomplish SIMD features. With data packed into one virtual *giant* register, more than one piece of data can be processed by a single instruction. For the DCT and IDCT implementation, we have performed 16 multiplications and 14 additions by using only five MMX instructions, thus reducing 35.5% of the clock cycles.

The gain in speedup is often circumvented by the overhead of data re-arrangement, data copying, data-type conversion, etc. (nonarithmetic operations) to suit the MMX instructions. Thus, the overall gain in performance may be restricted if the MMX instructions are not judiciously applied.

IV. Optimizations

The encoder is ehhanced with a variety of optimization techniques.

A. Algorithmic Optimization (AO)

We start with a fast search algorithm [6] that provides high speedup compared to full search block matching (FSBM). The algorithm uses reduced number of bits required for the motion vectors, yet maintaining the quality to an acceptable level. The algorithm partitions the search range into nested search zones, where the first zone is the innermost area of size 3×3 or 5×5 pixels. If the minimum MAD (mean absolute difference of a macroblock) can be found in the center, or if the matching error is less than a predefined threshold, the search procedure stops. Otherwise, the procedure continues to next consecutive zones. We use a threshold of 8 (threshold zero means full search) for zone 1, while our search range is $p = \pm 15$. However, we use a variation of [6] and, instead of making the threshold zero for zone 2, we rather use an even larger threshold of 12 (in the Ultra-1 implementation) or 16 (in the PC implementation) in order to maintain the obtained speedup. This way, we do not lose the advantages gained by the use of zone-based algorithm.

In order to fully exploit the advantages offered by the extended multimedia instruction sets, we make a modification in the block-matching process to further speed up the computation. We use a pixel-vector decimation technique similar to [13] so that only one 8-pixel vector is used for each row of pixels in the macroblock. Our approach is different from [13] in that, instead of subsampling pixels, we subsample vectors of pixels in both horizontal and vertical dimensions. The advantage of using 8-pixel vectors lies in their availability for VIS or MMX,[1] as the eight pixels are stored in a byte-aligned fashion in contiguous memory locations. This approach is illustrated in Fig. 2. If only the pixel vectors of pattern A are used for block matching, then the computation is reduced by a factor of four. However, since 75% of the pixel vectors do not enter into the matching computation, the use of this subsampling pattern alone can negatively affect the accuracy of motion vectors. To reduce this drawback, we use all four subsampling patterns, but only one at each search location and in a specific alternating (cyclic) manner. Therefore, if pattern A is used at the search location (x, y), then it is also used at locations $(x + 8i, y + 2j)$ for i, j integers within the search area; pattern B is used at locations $(x+8i, y+1+2j)$, pattern C at $(x+8+8i, y+1+2j)$ and pattern D at $(x+8+8i, y+2j)$. For each of the subsampling patterns, we obtain a motion vector that minimizes the *sum of absolute differences (SAD)* over the locations where the pattern is used. The minimum SAD, obtained from all four patterns, corresponds to the selected motion vector for the macroblock. By doing pixel-vector decimation, about 5% reduction in overall program running time is obtained.

The overall accuracy of DCT and IDCT is not affected by rounding off and truncations, which are intrinsic to the quantization process in video-coding applications. By exploiting this fact, we have designed a fast 8×8 DCT and IDCT algorithm (based on [18]) using VIS and MMX. The DCT/IDCT routines take an input block of 16-bit integers and deliver an output block of 16-bit integers. Since the input to the DCT routine is usually the difference between the current block and the reference block, the difference pixel can occupy 9 bits, and therefore, is represented as a 16-bit datum. We store four such 16-bit data

[1]In the case of MMX-based implementation, we compute the SAD based on 4-pixel vectors for block matching.

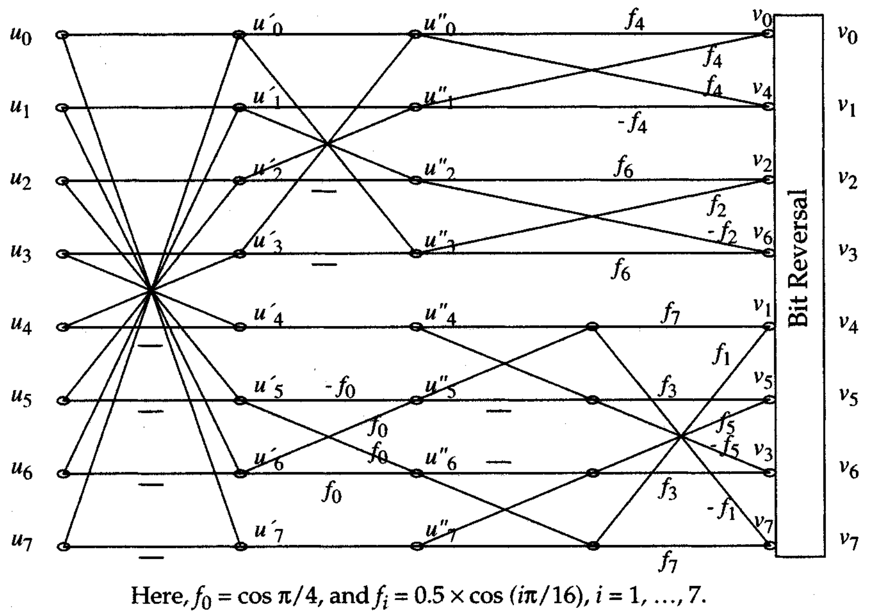

Bit Reversal
Here, $f_0 = \cos \pi/4$, and $f_i = 0.5 \times \cos (i\pi/16)$, $i = 1, \ldots, 7$.

Fig. 3. Signal flow graph for 8-point DCT.

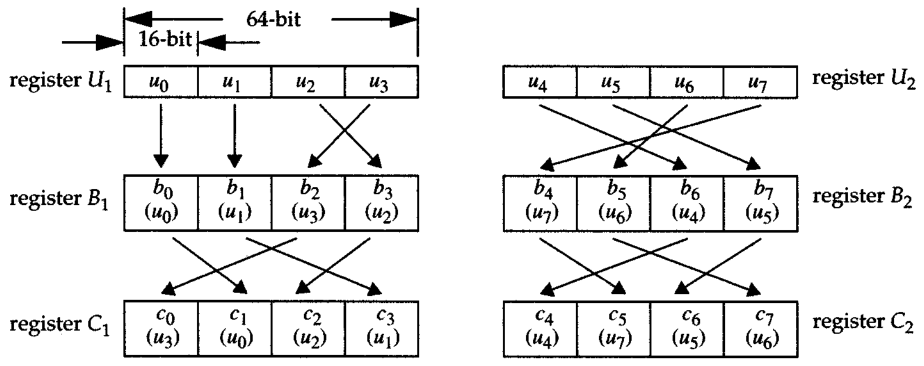


Fig. 4. Data rearrangement in 8-point DCT to facilitate VIS operations.

into a 64-bit register. We group these data elements such that DCT computation can be viewed as an SIMD parallel process. For instance, the transformation in the first stage of Fig. 3 can be written as

$$(u'_0,\, u'_1,\, u'_3,\, u'_2) = (u_0,\, u_1,\, u_3,\, u_2) + (u_7,\, u_6,\, u_4,\, u_5) \quad (1)$$

$$(u'_4,\, u'_7,\, u'_5,\, u'_6) = (u_3,\, u_0,\, u_2,\, u_1) - (u_4,\, u_7,\, u_5,\, u_6). \quad (2)$$

In order to perform the addition and subtraction in (1) and (2), we rearrange the input vector $[u]$ (residing in two registers U_1 and U_2) into registers B_1, B_2 and C_1, C_2, as shown in Fig. 4. This rearrangement is necessary to maintain the correspondence of data elements which are being operated on. By using 16-bit partitioned addition/subtraction of a 64-bit register on corresponding 16-bit data elements in registers B_1 and B_2, we obtain $(u'_0,\, u'_1,\, u'_3,\, u'_2)$ and $(u'_4,\, u'_7,\, u'_5,\, u'_6)$, respectively, which are stored in 64-bit registers D_1 and D_2, respectively.

Similarly, the transformation of the upper part of second stage can be written as

$$(u''_0,\, u''_1) = (u'_0,\, u'_1) + (u'_3,\, u'_2) \quad (3)$$

$$(u''_3,\, u''_2) = (u'_0,\, u'_1) - (u'_3,\, u'_2). \quad (4)$$

The values of (u''_0, u''_1) and (u''_3, u''_2) can be obtained by using 16-bit partitioned addition/subtraction on corresponding data elements of the upper and lower halves of D_1.

We have developed an efficient 32-bit × 32-bit multiplication strategy, where a pair of 16-bit × 16-bit multiplications are done in parallel. For instance, the upper part of the third stage of Fig. 3 requires three such multiplications, namely $(u''_0, u''_1) \times (f_4,\, f_4)$, $(u''_3,\, u''_2) \times (f_6,\, f_2)$, and $(u''_3,\, u''_2) \times (f_2,\, f_6)$. The upper and lower halves of the results (64 bits) of these multiplications can be added or subtracted using 32-bit partitioned addition/subtraction of 64-bit register on corresponding data elements in order to produce $[v_0,\, v_4,\, v_2,\, v_6]$. Similarly, $[v_1,\, v_5,\, v_3,\, v_7]$ can be obtained according to the transformation depicted in the lower part of stages 2–4 in Fig. 3.

B. Compiler Optimization

Most compilers come with optimizers that take advantage of sophisticated scheduling algorithms in order to perform software pipelining, for most efficient processing. Although we try to be as discreet as possible, some of the program optimization (mentioned in next section) may well be implicitly performed by the compiler optimizer.

For Ultra-1 based experiments, we have used the Sun Solaris C compiler (SC4.0) with the following flags setting: `xarch = v8plusa` (this flag defines the set of instruction the compiler should use), `xarch = ultra` (it defines the cache properties for use by the optimizer), `xdepend` (it enables all dependence based transformations), `fast` (it refers to the specification of a common set of performance options), and most importantly, `xO4` (it specifies that the compiler should generate optimized code at level 4). In addition, `xpg` is used for preparing object code to collect data for profiling using *gprof*.

For PC-based experiments, we have used the Microsoft Visual C++ compiler, with the following settings: `/Od` (it disables compiler optimization, we use it for the no optimization case), `/Ox` (it combines optimizing options to produce the fastest possible program), `/Ot` (it enables the compiler to reduce some C/C++ constructs to equivalent machine code), `/Oa` (it helps store variables in registers and perform loop optimization), `/Ow` (it ensures that after each function call, pointer variables must be reloaded from memory), `/Og` (it provides local and global optimizations, automatic register allocation and loop optimization), `/Oi` (it replaces some function calls with intrinsic functions, to avoid overhead of function calls), `/Op` (it improves consistency of floating points by disabling optimizations that could change floating-point precision), and `/Oy` (it omits frame pointers on the call stack and frees up one more register for storing frequently used variables and subexpressions). While using MMX instructions, we add the flag `/GM` in order to exploit further compiler optimization suitable for MMX instruction set. For a linker, optimization option `/OPT : REF` has been used.

C. Code Optimization

The following code-optimization techniques provide significant performance improvement [2], especially when the compiler optimizer fails to efficiently use the system resources.

Loop Unrolling: The H.263 encoder accesses data structures organized in matrices using loops. Some encoding functions require nested loops with several levels. Parallelism can be exploited by using pipelined access to such data structures by unrolling loops. Loop unrolling (LU) is the transformation of a loop so as to increase the loop body size and to decrease the number of iterations. This process may minimize both the number of load/store instructions by utilizing the CPU registers more efficiently, as well as data hazards arising from inefficient scheduling of instructions by the compiler optimizer. There may be two types of LU: Internal LU (ILU) and External LU (ELU). ILU consists of collapsing some iterations of the most internal loop into a larger and more complex statement requiring higher number of machine instructions, which can be more efficiently scheduled by the compiler optimizer. ELU consists of moving iterations from outer loops to inner loops, by using more registers in order to minimize the number of memory access inside the loop.

DTO: DTO is the choice of data types for the variables in the program critical path, which maximizes the performance of the different functional units, since the data types directly derived from the task definition may not yield the most efficient performance. We have used 16-bit integer values as the input and output of DCT and IDCT. In order to cope with the required floating-point operations in these functions, we have scaled the floating-point constants and allocated the precomputed constants to proper registers, instead of using the mixed-mode operations of integers and floating points.

Reduction of Redundant Operations: Divisions and multiplications are usually considered to be the most cycle-expensive operations. However, in most RISC processors, the integer (32 bit) multiply takes more cycles compared to the double (64 bit) multiply in terms of both instruction execution latency and instruction throughput [2]. In addition, floating-point divisions are less cycle-expensive compared to mixed-integer and floating-point divisions. Therefore, it is important to minimize the number of such arithmetic operations, especially inside a loop. Possible techniques include LU and DTO, while in some cases introduction of temporary variables (stored in registers) can provide noticeable performance improvement. We have used such techniques for the quantization module of our implementation.

V. Experimental Results and Discussion

In this section, we present the experimental results and comparisons of our implementation on two platforms, namely the PCs and the workstation and compare the corresponding performances.

A. Test Video Streams

We have used nine video streams of QCIF resolution: *Claire*, *Grandma*, *Miss America*, *Salesman*, *Mother and Daughter*, *Trevor*, *Car Phone*, *Suzie*, and *Foreman*. These video sequences represent various type of motion, both in terms of motion in scene content and camera motion. The variety of motion makes the complexity of the ME process to be different for each video sequence, while the time to calculate the motion vectors are also of wide variation range. Being the most time-consuming encoder module, the performance of ME affects the total encoder running time. We may divide the above nine video sequences into two major categories, depending on the difficulty (and therefore time taken) to compute the motion vectors: sequences with *slow* motion (SM) and sequences with *fast* motion (FM). The sequences *Claire*, *Grandma*, *Miss America*, and *Salesman* may fall into the SM category while *Mother and Daughter*, *Trevor*, *Car Phone*, *Suzie*, and *Foreman* fall into the FM category.

B. Analysis of Computational Requirements of Modules

In this section we present an analysis of the execution profile of our H.263 encoder using the GNU *gprof* profiler. Our implementation of optimized software-based H.263 encoder is based on the Telenor's H.263 video encoder [23]. Since the coding of I-frame is performed only once (for the first frame) and does not require expensive operations like motion estimation, we restrict our analysis to the coding of one video frame as a P-frame or two video frames as PB-frames. This analysis shows that 97.3% of the program running time is spent on the principal encoding function. The computation requirements of various functional modules within this principal encoding function are analyzed below in details. Since insignificant amount of time is spent on

TABLE I
TIME REQUIREMENTS FOR MOST COMPUTATION-INTENSIVE FUNCTIONAL MODULES, AS A PERCENTAGE OF THE PRINCIPAL ENCODING FUNCTION

Module	Implementation on SUN UltraSPARC-1			Implementation on Intel 233MHz PII PC		
	No Optimization		All Optimizations	No Optimization		All Optimizations
	All Optional Modes	No Optional Mode	No Optional Mode	All Optional Modes	No Optional Mode	No Optional Mode
Motion Estimation	64.4	85.4	29.7	66.49	84.57	28.14
Prediction of B-picture	11.2	-	-	12.17	-	-
Macroblock (MB) Encoding	9.8	7.7	33.6	8.97	7.58	16.92
Prediction of P-picture	3.7	1.8	4.3	1.91	1.63	5.35
MB Reconstruction (P)	3.2	0.7	2.1	1.77	0.75	3.89
Interpolation of Image	1.2	1.1	4.4	1.2	1.26	5.35
MB Reconstruction (B)	1.1	-	-	1.40	-	-
Clipping	1.0	0.7	1.5	1.54	0.94	3.89
Macroblock Decoding	0.8	0.8	10.0	1.22	1.33	13.13
Coding of quantized coeff.	0.3	0.3	5.8	0.00	0.11	1.73

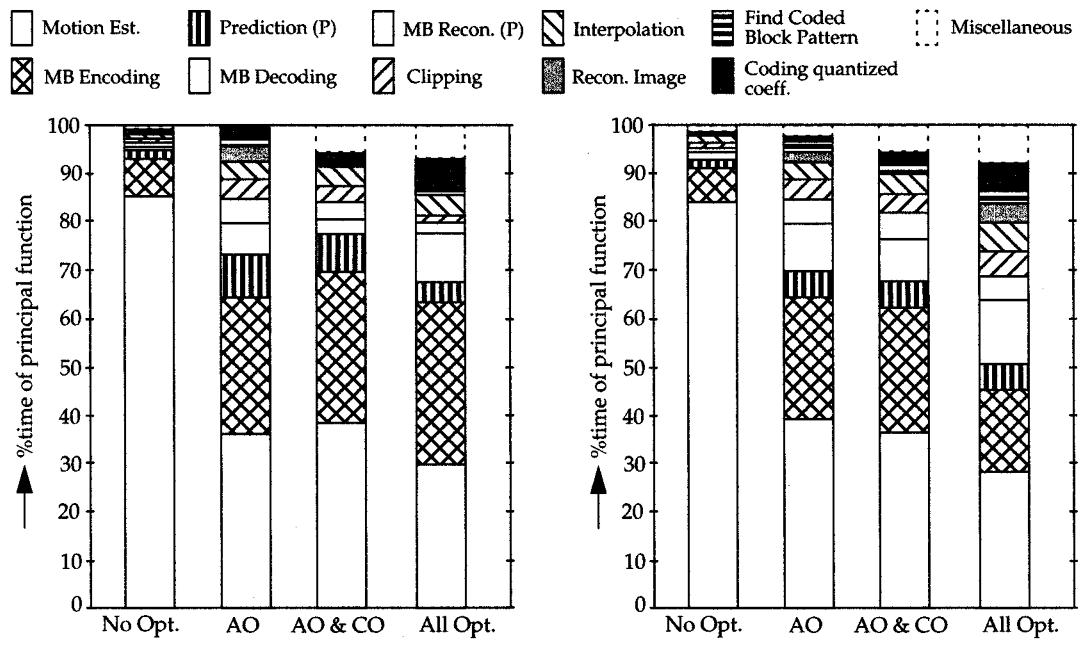


Fig. 5. Effect of optimizations.

performing input (about 2%) and video quality measurement in terms of PSNR (0.6%)—these functions are not probed any further.

Table I shows the breakdown of the execution time of the principal encoding function into various constituent modules. Together, they require 92%–98% of the execution time of the principal encoding function. It may be observed that, for the no optimization case, *motion estimation* is the most time consuming module, followed by *macroblock encoding* (which involves DCT and quantization) and motion-compensated prediction. Unlike [20], which reports a high percentage of *macroblock decoding* time, our approach adopts finding IDCT for only the nonzero elements, thereby reducing the *macroblock decoding* time considerably.

With the application of all the optimizations discussed in Section IV, the execution profile of our program changes noticeably. With optimizations, the computational requirement of *motion estimation* is reduced to almost one third (in terms of percentage points). Therefore, percentage execution time of other functional modules increases. Fig. 5 illustrates this effect.

C. Performance of the H.263 Encoder

The reported results were obtained using the first 100 frames of each video sequence. The encoding rates are given in frames per second. The reference frame rate was kept at 30 frames/s while the input original sequence frame rate was assumed to be 30 frames/s. As an encoding output parameter, we used both

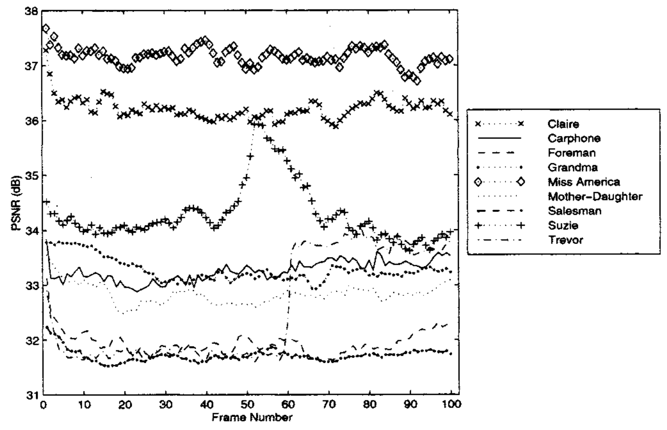


Fig. 6. Luminance PSNR with no optional mode.

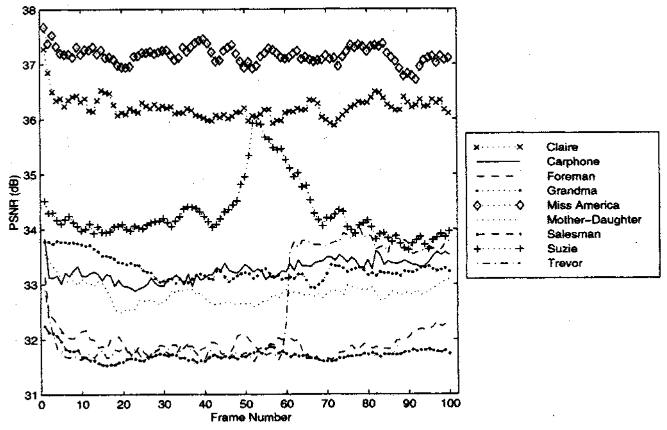


Fig. 8. Luminance PSNR with *arithmetic coding* mode.

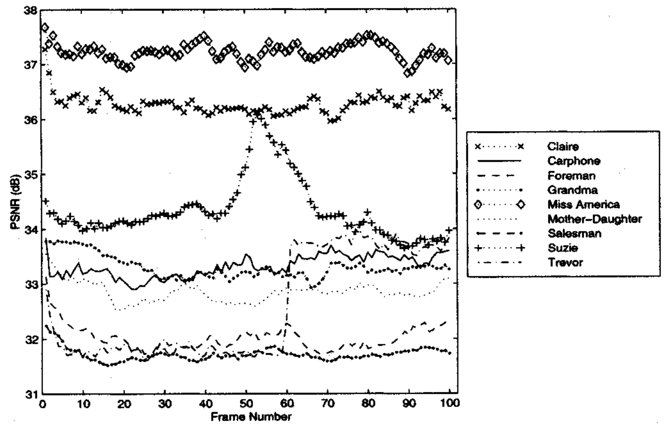


Fig. 7. Luminance PSNR with *UMV* mode.

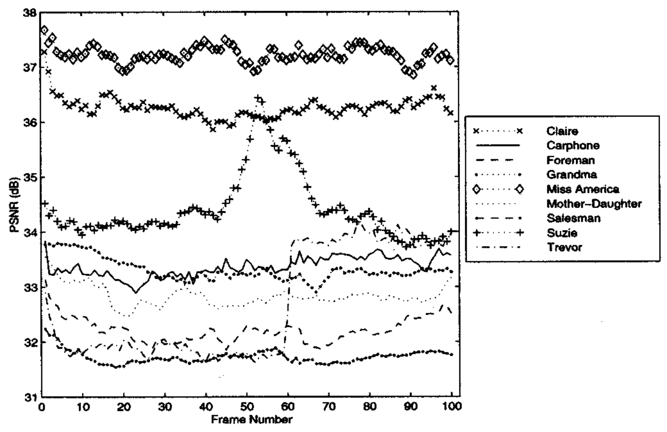


Fig. 9. Luminance PSNR with *advanced prediction* mode.

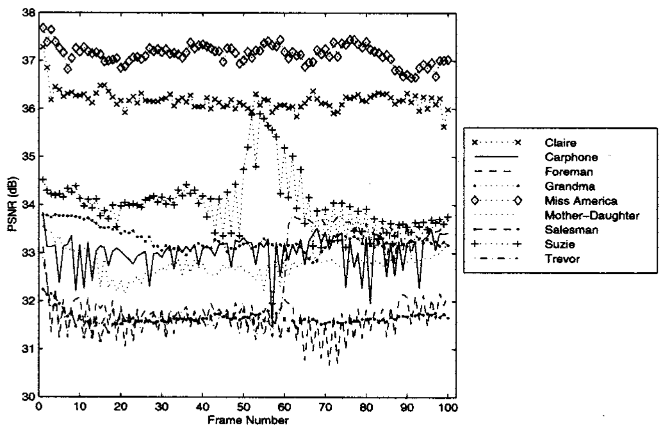


Fig. 10. Luminance PSNR with *PB-frames* mode.

variable bit rate [with encoding frame rate[2] at 30 frames/s, QP at 10, and constant bit rate (with variable QP and encoding frame rate, but bit rate fixed at 28.8 kbits/s). To measure the actual program running time, we used available library functions (`hrtime` and `clock`), which are accurate up to microseconds. The timing results were averaged over 100 runs. In addition to performing the experiments on a 167-MHz Sun Ultra-1, we performed the same experiments with two PCs: a 233-MHz Pentium II (PC PII) and a 600-MHz Pentium III (PC PIII).

Figs. 6–11 depict the luminance PSNR under different optional modes with variable bit rates. From these figures, it is evident that PSNR does not change noticeably due to the incorporation of various optional modes. Subjective quality, as observed, also remains the same. Therefore, with the same quality, our choice of encoder is concentrated on the encoder speed.

Fig. 12 shows the luminance PSNR with no optional mode at a constant bit rate of 28.8 kbits/s. The experiment was still performed with 100 frames, but the encoding frame rate was variable, in order to meet the constraint of fixed bit rate. Therefore, 25–29 frames of the sequences were coded, depending on the complexity of (and therefore bit spent to encode) the sequence. The mean QP is 5.68–17.88 while the mean encoded frame rate (in terms of the ratio of allocated bit rate and actual number of bits to encode the frame) is 9.44–9.88 frames/s. It is evident from this figure that, with a constant bit rate, the SM sequences yield very good quality while the quality of the FM sequences is still acceptable.

[2]*Endcoding frame rate* refers to the ratio of the allocated bit rate and the actual number of bits to encode the frames. It depends on the quantization parameter (QP) and the number of frames. Note that this frame rate is different from the *frame encoding speed*, which is a measure of the actual running time of the program in terms of frames per second, and depends on the computational and programming complexity.

Table II shows the frame encoding speed in frames/s for our H.263 video encoder. These results involve no explicit optimization. Although we disabled explicit compiler optimization for

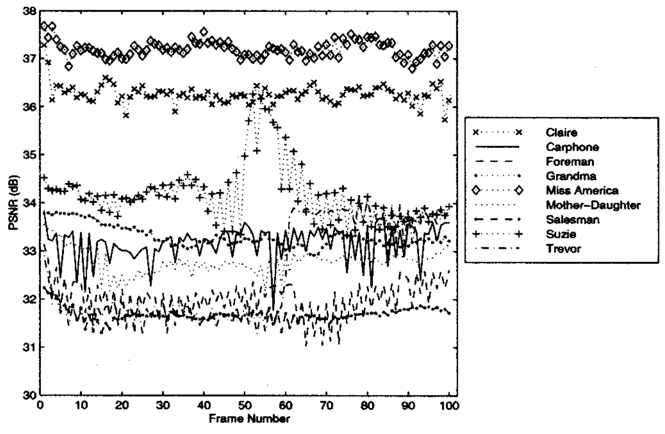


Fig. 11. Luminance PSNR with all optional modes.

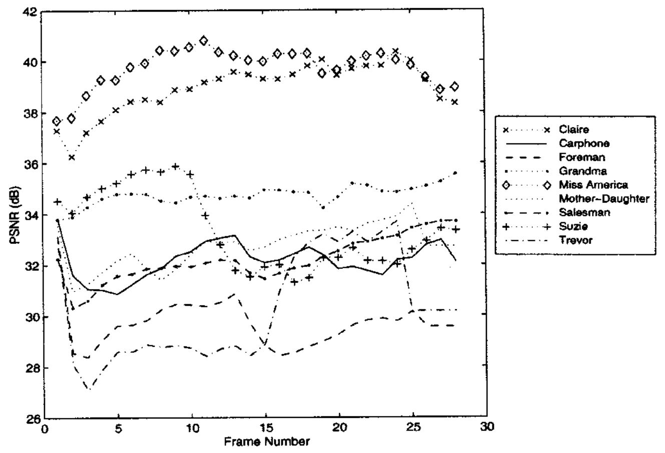


Fig. 12. Luminance PSNR with no optional mode at 28.8 kbits/s.

the PC by using /Od switch for the Microsoft Visual C++ compiler, the compiler uses some intrinsic optimizations. As a result, the PC version of the encoder yields faster encoding speed compared to the Ultra-1 workstation. It may be observed from this table that the use of optional modes considerably slow down the encoding speed. Table III shows the encoding speed with AO performed on motion estimation, DCT and IDCT. The effect of using AO is discussed in Section V-D. The multimedia instruction sets are not used in these cases.

Table IV shows the encoding speed with AO and compiler optimization. Further fine tuning in optimization is done by reducing some cycle-expensive operations, and including some code optimization, especially LU and DTO. The effect is discussed in Section V-D.

Table V includes the H.263 encoding speed with all the optimizations. The encoder achieved a maximum frame-encoding speed of 18.12 frames/s using the PC PII and 12.17 frames/s using the Ultra-1 workstation. On the Ultra-1, the mean frame encoding speed with no optional mode is 11.28 frames/s. Using the PB-frames, SAC, UMV, and advanced prediction modes, the average frame encoding speed is 11.27, 10.91, 8.98, and 7.15 frames/s, respectively. Using all the optional modes, the average frame encoding speed goes down to 6.72 frames/s. On the PC PII, the mean frame encoding speed is 16.05 frames/s without optional modes. Using the SAC and the PB-frames modes, the average frame encoding speed is 15.78 and 15.04 frames/s, respectively. However, with use of the UMV mode, the advanced prediction mode and all optional modes yielded average frame encoding speeds of 14.26, 11.44, and 10.44 frames/s, respectively.

Table VI shows the percentage loss in encoding speed using various optional modes compared to no optional mode. The significance of these results is discussed in the next section.

Table VII shows the average luminance PSNR without optimization, which does not increase significantly with the use of optional modes.

In Table VIII, perceptible changes in PSNR are not due to optimizations. This fact is further highlighted by the subjective judgment of visual quality. Under some test conditions, however, the use of optional modes may increase the PSNR by about 1 dB [5].

Table IX shows the average obtained bit rate for the no optimization case, with the quantization QP fixed at a value of 10. Values shown in parentheses represent bit rates for interpicture only, while those without parentheses represent bit rates including intra-coded pictures. Different test sequences, having a variety of motion involved, require different bit rates (for the QP fixed at 10), ranging from 18 to 113 kbits/s for various optional modes. The bit rate could be fixed to a particular value (say, 64 or 28.8 kbits/s). However, in that case, sequences with complex motion (FM sequences) would take more encoding time and the quality would be poorer as well. This effect is shown in Fig. 12.

Table X shows the average bit rate for the optimized case with the QP fixed at 10. For the SUN Ultra-1 implementation, only a small increase in bit rate is observed compared to without optimization. This is due to the incorporation of fast search instead of FSBM. Even for the PC-based implementation, the increase in bit rate is not significant, and the implementation is still applicable with currently available modems.

Table XI depicts a comparison of encoding speed using two different PC platforms: the 233-MHz Pentium II (PC PII) and the 600-MHz Pentium III (PC PIII). In this comparison, we only consider our H.263 encoder without optional modes. With an increase in clock speed (2.58 times), the PC PIII consistently gives higher encoding speed (2.49–2.57 times). The bit rates are variable, but the average bit rate (while QP is fixed at 10) for both the PCs is almost the same.

1) Effect of Coding Options: From Table VI, we observed that the use of optional modes obviously increases the complexity of the encoder, hence increasing the overall program running time. However, these optional modes may be useful for higher bit rates. Since we deal with QCIF sizes of video frames and more common bit rates pertinent to most of the H.263-based applications, the use of the PB-frames mode may be a better choice, which provides a balance between encoding speed and bit rate.

For the Ultra-1 implementation, the use of the PB-frames mode incurs very low expense (about 2%) in additional time requirement for the SM type of video sequences. It is interesting to note that there is an actual gain (1%–4%) in the encoding speed using the PB-frames mode for the FM type of video sequences. The use of only the PB-frames mode keeps almost the same quality with 13%–28% less bit rate. This finding affirms

TABLE II
ENCODING SPEED IN FRAMES/S (NO OPTIMIZATION)

Sequences	No Optional Mode		With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	0.57	2.11	0.54	1.96	0.57	2.10	0.44	1.67	0.53	1.94	0.47	1.73
Grandma	0.49	1.66	0.43	1.46	0.49	1.65	0.36	1.32	0.45	1.57	0.40	1.41
Miss America	0.46	1.54	0.40	1.41	0.45	1.52	0.34	1.28	0.35	1.45	0.32	1.25
Salesman	0.52	1.74	0.47	1.65	0.52	1.72	0.39	1.42	0.49	1.70	0.44	1.47
Mother and Daughter	0.52	1.76	0.47	1.66	0.51	1.74	0.39	1.41	0.42	1.73	0.38	1.46
Trevor	0.50	1.67	0.46	1.46	0.49	1.65	0.38	1.33	0.42	1.58	0.38	1.42
Car Phone	0.53	1.90	0.47	1.73	0.53	1.88	0.40	1.59	0.39	1.85	0.34	1.59
Suzie	0.45	1.57	0.39	1.45	0.45	1.55	0.33	1.30	0.36	1.49	0.32	1.47
Foreman	0.51	1.76	0.49	1.67	0.51	1.74	0.41	1.40	0.40	1.72	0.37	1.44

TABLE III
ENCODING SPEED IN FRAMES/S (WITH ALGORITHMIC OPTIMIZATION ONLY)

Sequences	No Optional Mode		With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	1.66	4.83	1.57	4.43	1.67	4.82	1.09	3.49	1.30	3.89	1.06	3.33
Grandma	1.65	4.40	1.57	4.00	1.66	4.38	1.09	3.19	1.28	3.69	1.05	3.02
Miss America	1.58	4.69	1.53	4.21	1.63	4.67	1.07	3.35	1.27	3.78	1.02	3.12
Salesman	1.60	4.26	1.56	3.87	1.64	4.24	1.08	3.12	1.24	3.43	1.04	2.98
Mother and Daughter	1.57	4.15	1.51	3.91	1.57	4.12	1.05	3.20	1.23	3.58	1.00	3.09
Trevor	1.47	3.92	1.41	3.84	1.47	3.89	1.01	3.08	1.10	3.61	0.92	2.81
Car Phone	1.51	3.98	1.46	3.68	1.54	3.95	1.04	3.06	1.28	3.36	0.97	2.79
Suzie	1.49	4.04	1.47	3.89	1.53	4.01	1.04	3.11	1.16	3.66	0.96	3.01
Foreman	1.50	3.67	1.45	3.37	1.49	3.64	1.03	2.86	1.22	3.07	0.97	2.70

TABLE IV
ENCODING SPEED IN FRAMES/S (ADDING COMPILER OPTIMIZATION)

Sequences	No Optional Mode		With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	10.65	10.65	8.10	9.55	10.56	10.48	6.72	8.33	8.96	8.97	5.13	8.05
Grandma	10.44	10.21	8.11	9.06	10.42	10.25	6.45	7.96	8.33	9.06	4.74	7.18
Miss America	10.35	10.73	8.09	9.27	10.22	10.59	6.46	8.16	8.64	9.05	5.34	7.58
Salesman	10.11	9.85	7.75	8.81	10.12	9.84	6.37	7.78	8.49	8.81	5.14	7.35
Mother and Daughter	9.72	10.10	7.65	9.00	9.32	10.02	6.01	7.98	8.00	9.01	5.25	7.11
Trevor	9.09	8.85	7.21	7.88	8.64	8.80	5.90	7.23	7.26	8.06	4.57	6.92
Car Phone	9.51	9.30	7.49	7.77	9.12	8.46	6.06	7.27	8.10	7.94	4.93	6.77
Suzie	9.46	9.93	7.53	8.86	8.98	9.75	6.06	7.63	7.62	8.36	5.06	7.05
Foreman	9.22	8.11	7.60	7.27	8.83	7.99	6.02	6.94	8.00	7.44	4.98	6.77

the fact that in H.263, B-pictures are computationally much less expensive than P-pictures; because the motion estimation can be done in a much smaller area, fewer blocks are coded, and on average, fewer coefficients are transmitted per block.

TABLE V
ENCODING SPEED IN FRAMES/S (WITH ALL OPTIMIZATIONS)

Sequences	No Optional Mode		With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	12.17	18.12	9.23	15.81	11.99	17.93	7.61	12.33	11.85	16.83	6.91	11.38
Grandma	11.90	18.06	9.16	15.79	11.71	17.84	7.25	12.45	11.68	16.83	6.93	11.32
Miss America	11.88	17.30	9.23	15.26	11.45	17.04	7.32	11.98	11.59	16.12	6.96	10.96
Salesman	11.66	17.45	8.86	15.40	11.59	17.37	7.17	12.23	11.55	16.31	6.51	11.12
Mother and Daughter	11.11	15.53	8.98	13.86	10.80	15.21	7.05	11.17	11.20	14.69	6.74	10.35
Trevor	10.93	14.69	8.86	13.43	10.43	14.38	7.06	10.91	11.05	14.02	6.48	9.78
Car Phone	10.89	14.74	8.97	13.13	10.43	14.45	6.97	10.71	10.98	13.82	6.76	9.85
Suzie	10.86	14.76	8.92	13.47	10.44	14.50	7.12	11.02	11.05	13.96	6.72	9.95
Foreman	10.10	13.78	8.61	12.22	9.38	13.31	6.75	10.20	10.49	12.76	6.48	9.26

TABLE VI
PERCENTAGE LOSS IN ENCODING SPEED DUE TO VARIOUS OPTIONAL MODES

Sequences	With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	24.16	12.75	1.48	1.05	37.47	31.95	2.63	7.12	43.22	37.20
Grandma	23.03	12.57	1.60	1.22	39.08	31.06	1.85	6.81	41.76	37.32
Miss America	22.31	11.79	3.62	1.50	38.38	30.75	2.44	6.82	41.41	36.65
Salesman	24.01	11.75	0.60	0.46	38.51	29.91	0.94	6.53	44.17	36.28
Mother and Daughter	19.17	10.75	2.79	2.06	36.54	28.07	-0.81	5.41	39.33	33.35
Trevor	18.94	8.58	4.57	2.11	35.41	25.73	-1.10	4.56	40.71	33.42
Car Phone	17.63	10.92	4.22	1.97	36.00	27.34	-0.83	6.24	37.92	33.18
Suzie	17.86	8.74	3.87	1.76	34.44	25.34	-1.75	5.42	38.12	32.59
Foreman	14.75	11.32	7.13	3.41	33.17	25.98	-3.86	7.40	35.84	32.80

TABLE VII
AVERAGE LUMINANCE PSNR IN DECIBELS (WITHOUT OPTIMIZATION)

Sequences	No Optional Mode		With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	36.2127	36.20	36.2701	36.26	36.2127	36.20	36.2541	36.24	36.1825	36.18	36.2703	36.28
Grandma	33.2771	33.27	33.3017	33.30	33.2771	33.27	33.3125	33.31	33.2760	33.26	33.3051	33.30
Miss America	37.1663	37.16	37.2293	37.22	37.1663	37.16	37.2249	37.22	37.1377	37.14	37.2013	37.23
Salesman	31.7168	31.71	31.7154	31.71	31.7168	31.71	31.7276	31.72	31.6297	31.65	31.6837	31.71
Mother and Daughter	32.8343	32.87	32.8511	32.88	32.8343	32.87	32.8334	32.89	32.7679	32.76	32.7844	32.90
Trevor	32.5736	32.60	32.5937	32.61	32.5736	32.60	32.6775	32.63	32.3980	32.69	32.5378	32.72
Car Phone	33.2553	33.25	33.3164	33.31	33.2553	33.25	33.3642	33.36	33.0275	33.19	33.1712	33.34
Suzie	34.2912	34.35	34.3428	34.46	34.2912	34.35	34.4221	34.48	34.0425	34.51	34.1903	34.55
Foreman	31.9361	31.93	32.0091	32.00	31.9361	31.93	32.2133	32.21	31.6875	31.92	32.0171	32.28

The use of the SAC mode is the next less expensive mode. In our experiments, the additional cost appears to be about 1%–7% compared to the no-optional mode. In this case, the bit rate is reduced by 3%–6%. However, the (UMV) mode accounts for much more encoding time, and the additional cost is about 22%–24% for the SM type of sequences and 15%–19% for the

TABLE VIII
AVERAGE LUMINANCE PSNR IN DECIBELS (WITH ALL OPTIMIZATIONS)

Sequences	No Optional Mode		With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	36.0371	35.95	36.0934	35.98	36.0371	35.95	36.1092	36.00	36.0226	35.96	36.1239	36.00
Grandma	33.2762	33.17	33.3011	33.17	33.2762	33.17	33.3037	33.17	33.2728	33.17	33.3124	33.17
Miss America	37.0508	36.50	37.1291	36.51	37.0508	36.50	37.1083	36.48	37.0265	36.50	37.0892	36.48
Salesman	31.7032	31.66	31.7034	31.66	31.7032	31.66	31.7164	31.69	31.6417	31.66	31.6722	31.69
Mother and Daughter	32.8342	32.77	32.8661	32.79	32.8342	32.77	32.8327	32.80	32.7562	32.82	32.7953	32.87
Trevor	32.4686	32.56	32.5459	32.58	32.4686	32.56	32.6044	32.58	32.3008	32.62	32.4747	32.69
Car Phone	33.1477	33.04	33.2362	33.10	33.1477	33.04	33.2464	33.10	32.9221	33.04	33.0792	33.10
Suzie	34.1653	34.33	34.2567	34.40	34.1653	34.33	34.2908	34.42	33.9285	34.47	34.0629	34.51
Foreman	31.8254	31.59	31.9107	31.63	31.8254	31.59	31.9476	31.65	31.6823	31.59	31.9092	31.65

TABLE IX
AVERAGE BIT RATE IN KBITS/S (WITHOUT OPTIMIZATION)

Sequences	No Optional Mode		With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	26.98 (22.26)	26.93 (22.26)	26.92 (22.20)	26.89 (22.22)	26.04 (22.12)	26.01 (22.13)	26.31 (21.59)	26.26 (21.58)	21.44 (16.67)	21.45 (16.72)	19.34 (15.35)	19.36 (15.41)
Grandma	25.12 (20.55)	25.01 (20.49)	25.01 (20.44)	24.90 (20.38)	24.44 (20.20)	24.34 (20.15)	24.54 (19.97)	24.40 (19.88)	19.46 (14.84)	19.42 (14.85)	18.55 (14.25)	18.49 (14.23)
Miss America	27.42 (24.49)	27.51 (24.61)	27.15 (24.22)	27.29 (24.39)	26.50 (23.90)	26.60 (24.02)	26.52 (23.58)	26.61 (23.70)	20.04 (17.04)	20.14 (17.17)	18.39 (15.71)	18.51 (15.85)
Salesman	40.23 (33.85)	40.04 (33.72)	40.25 (33.87)	40.07 (33.75)	39.06 (33.28)	38.89 (33.16)	38.74 (32.35)	38.57 (32.24)	32.31 (25.85)	32.23 (25.83)	29.94 (24.07)	29.85 (24.03)
Mother and Daughter	58.83 (54.17)	58.80 (54.15)	57.66 (52.99)	57.63 (52.95)	56.02 (51.84)	55.97 (51.81)	57.24 (52.56)	57.20 (52.50)	45.79 (41.00)	45.75 (40.95)	42.33 (38.01)	42.30 (38.00)
Trevor	97.01 (90.79)	97.00 (92.09)	94.83 (88.58)	94.80 (88.76)	93.11 (87.66)	93.17 (87.96)	90.60 (84.31)	91.33 (84.82)	78.41 (71.99)	79.01 (72.14)	70.52 (64.85)	70.00 (64.51)
Car Phone	82.75 (76.99)	82.60 (76.89)	81.25 (75.75)	81.35 (75.63)	78.73 (73.76)	78.59 (73.67)	78.70 (72.89)	78.49 (72.74)	72.48 (66.61)	72.43 (66.62)	65.10 (59.99)	65.07 (60.02)
Suzie	65.08 (61.51)	65.00 (61.37)	62.68 (59.08)	62.59 (59.00)	61.07 (57.96)	61.01 (57.92)	61.00 (57.39)	60.98 (57.76)	53.03 (49.34)	52.99 (49.15)	46.78 (43.53)	46.75 (43.50)
Foreman	113.08 (107.22)	112.68 (106.87)	102.23 (96.25)	102.03 (96.11)	106.92 (101.78)	106.53 (101.44)	97.33 (91.31)	97.02 (91.06)	81.47 (75.29)	81.84 (75.36)	63.78 (58.21)	63.86 (58.35)

FM type of sequences. Using the UMV, there is little gain in quality with almost the same bit rate. This result suggests that, if there is little or no motion at or near the boundary region (as in the case of the SM sequences), the overhead due to UMV is much higher compared to those with faster or complex motion (the FM sequences).

The advanced prediction mode is the most expensive mode that slows down the encoder by 33%–39%. Using this mode, we obtained almost the same quality while the savings in bit rate is less than 10%. The reason is that the overhead of computing four 8×8 motion vectors instead of one 16×16 motion vector is obviously higher. With this mode, 8×8 motion vectors are chosen for 65%–75% of the macroblocks. Overall, using all the modes simultaneously, the encoder runs at about 36%–44% slower speed compared to no optional mode.

In the case of the PC PII-based implementation, SAC mode proves to be the most efficient among the optional modes, requiring only about 0.5%–3.5% more encoding time than no optional mode at almost the same bit rate. The PB-frames mode is the next, which requires about 4.5%–7.5% more encoding time, but saves about 3.5% of the bits. The use of the UMV mode is 8%–13% slower, while the advanced prediction mode needs 25%–32% more encoding time. Together, the use of all optional modes slows down the encoder by about 33%–38%.

D. Effect of Optimizations

Fig. 5 shows the percentage of encoding time for various modules. From our experimental results as shown in Table III, the use of AO alone gives about three-fold speedup in encoding

TABLE X
AVERAGE BIT RATE IN KBITS/S (WITH ALL OPTIMIZATIONS)

Sequences	No Optional Mode		With UMV		With SAC		With Advanced Prediction		With PB-frames		With All Optional Modes	
	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII	Ultra	PC PII
Claire	26.94 (22.22)	28.53 (23.82)	26.62 (21.89)	28.40 (23.69)	26.00 (22.08)	27.58 (23.64)	25.92 (21.19)	27.56 (22.84)	21.28 (16.50)	28.53 (23.82)	19.08 (15.09)	26.67 (22.72)
Grandma	25.02 (20.46)	26.32 (21.72)	24.93 (20.36)	26.28 (21.67)	24.36 (20.12)	25.60 (21.32)	24.46 (19.89)	25.77 (21.16)	19.58 (14.96)	26.32 (21.72)	18.45 (14.15)	25.14 (20.86)
Miss America	27.36 (24.44)	31.92 (29.04)	26.97 (24.04)	32.52 (29.64)	26.45 (23.85)	31.84 (29.28)	26.44 (23.50)	32.22 (29.34)	21.26 (18.28)	31.92 (29.04)	19.42 (16.74)	31.22 (28.65)
Salesman	40.51 (34.14)	42.02 (36.62)	40.59 (34.21)	42.02 (36.62)	39.37 (33.59)	41.86 (36.05)	38.86 (32.47)	41.78 (35.37)	32.64 (26.18)	40.02 (35.62)	30.48 (24.61)	37.82 (32.79)
Mother and Daughter	58.94 (54.28)	59.87 (55.01)	57.82 (53.15)	59.88 (55.01)	56.17 (51.99)	57.05 (52.36)	57.15 (52.47)	58.23 (53.52)	47.25 (42.47)	55.13 (50.55)	43.43 (39.12)	53.26 (49.61)
Trevor	97.54 (91.32)	98.02 (93.44)	95.18 (88.94)	96.42 (91.08)	93.66 (88.22)	94.75 (89.64)	91.68 (85.39)	94.14 (89.57)	81.59 (75.20)	90.11 (85.46)	71.53 (65.86)	87.32 (82.29)
Car Phone	85.66 (79.92)	98.46 (93.90)	82.40 (76.63)	96.39 (92.81)	81.58 (76.64)	94.60 (91.81)	80.92 (75.14)	94.09 (90.49)	76.76 (70.94)	98.46 (93.90)	67.10 (62.01)	90.40 (86.57)
Suzie	71.97 (68.47)	72.13 (68.22)	64.82 (61.25)	67.72 (62.07)	67.68 (64.63)	68.86 (63.55)	68.43 (64.90)	69.93 (65.21)	61.27 (57.66)	67.45 (62.33)	51.19 (47.98)	65.44 (60.12)
Foreman	116.42 (110.59)	128.30 (122.94)	102.24 (96.27)	116.96 (111.49)	110.10 (104.99)	120.72 (116.06)	102.78 (96.81)	115.67 (110.19)	99.93 (93.94)	128.30 (122.94)	73.46 (67.99)	108.62 (103.84)

TABLE XI
COMPARISON OF THE H.263 ENCODER (WITH ALL OPTIMIZATIONS) ON PCs

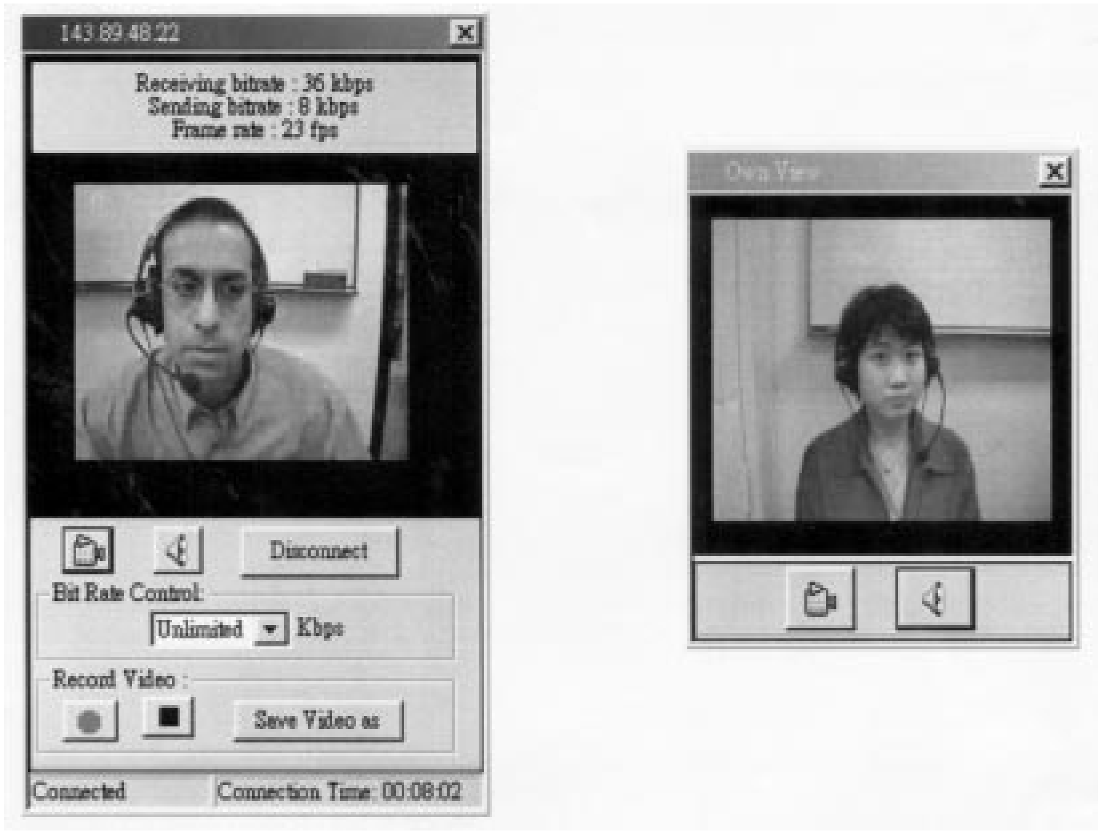
Sequence	Average bit rate (kb/s)		Encoding speed (frames/sec.)		Speedup
	PC PII	PC PIII	PC PII	PC PIII	
Claire	28.53 (23.82)	26.97 (22.25)	18.12	45.68	2.52
Grandma	26.32 (21.72)	25.75 (21.10)	18.06	45.21	2.50
Miss America	31.92 (29.04)	31.23 (29.15)	17.30	44.50	2.57
Salesman	42.02 (36.62)	41.12 (36.16)	17.45	43.61	2.49
Mother and Daughter	59.87 (55.01)	59.11 (54.51)	15.53	39.66	2.55
Trevor	98.02 (93.44)	97.48 (93.02)	14.69	37.31	2.54
Car Phone	98.46 (93.90)	97.56 (93.04)	14.74	37.57	2.55
Suzie	72.13 (68.22)	72.07 (68.19)	14.76	37.78	2.56
Foreman	128.30 (122.94)	128.05 (122.55)	13.78	35.27	2.56

speed, compared to the no-optimization case. From Table IV, which involves the AO, compiler optimization, and reduction of some cycle-expensive operations, the additional optimizations provide a 6–7 times more improvement in encoding speed. Further inclusion of VIS provides a speedup of about 20% on the overall program running time. All in all, with all the optimizations, about 20–26 times improvement in speed is observed with no optional mode, while about 15–22 times speedup is gained with all optional modes. In summary, we make the following observations.

1) Optimizations at algorithmic level are the most important consideration for computation-intensive functional modules, particularly for full-search block matching, DCT, IDCT, quantization, and inverse quantization.
2) Although some of the loop unrolling (LU) may be performed by the compiler optimizer, LU is an effective optimization technique. It can be applied to loop-intensive functional modules such as motion estimation and motion compensated prediction.
3) DTO provides improved performance when complicated type conversions must be handled. For instance, DCT includes such operation in order to take advantage of the 64-bit registers, and DTO is very useful in such cases.
4) Motion estimation, motion compensated prediction, DCT, and IDCT are the functional modules that deal with regular data structures, and are amenable to VIS/MMX-based optimization.

E. A Videophone Application

We have built a videophone using our optimized H.263 video encoder. We use QCIF resolution of video, captured via a video camera that may use the USB port of the PC. The videophone displays both the called party and the calling party on separate windows, while the PC runs both the H.263 encoder and



Receiving bitrate : 36 kbps
Sending bitrate : 8 kbps
Frame rate : 23 fps

Bit Rate Control: Unlimited Kbps

Connected
Connection Time: 00:08:02

Fig. 13. Scene from the videophone application.

decoder. In addition to video, we also use an audio codec (for which an additional bandwidth of about 10 kbits/s is necessary, but this issue is not further discussed in this paper). The videophone reports the frame rate and the bit rates in real-time. For a typical videophone, the motion involved in the scene is usually slow, allowing lower bit rate with good quality. An instance of the videophone, as shown in Fig. 13, reveals that the bit rate of the sending and receiving bit streams are 8 and 36 kbits/s, respectively, with a display frame rate of 23 frames/s. Although the measured encoding speed in this case is 44.8 frames/s, due to a constraint in available bandwidth, capture frame rate, etc., an overall throughput of only 23 frames/s has been achieved. The bit-rate control option is set to "unlimited" (i.e., variable) with a fixed QP of 10. However, we can also keep the bit rate at a constant level (e.g., at 64 kbits/s), while allowing the QP to change. For a fixed bit rate, if fast motion is involved, the visual quality will be poorer. However, for slow motion, a bit-rate ceiling of 28.8 kbits/s is usually sufficient to yield good to excellent quality.

VI. Conclusion

We have presented the implementation of an optimized software-based real-time H.263 video encoder. In order to achieve enhanced performance, various software optimizations and low-level machine primitives such as VIS and MMX are exploited. We have achieved a video-encoding speed which is sufficient for most GSTN-based applications. In addition, we have presented a discussion about the optimal choice of the encoder. It has been found that the use of PB-frames mode is a good choice for encoding, which provides a balance between PSNR, bit rate, and encoding speed. Our present work focuses on the incorporation of new and improved algorithms for various encoder modules, which can be easily used replacing the existing algorithms, without altering the backbone of our implementation.

References

[1] S. M. Akramullah, I. Ahmad, and M. L. Liou, "A software-based H.263 video encoder using a cluster of workstations," *Proc. SPIE*, vol. 3166, pp. 266–273, 1997.

[2] P. Baglietto, M. Maresca, M. Migliardi, and N. Zingirian, "Image processing on high-performance RISC systems," *Proc. IEEE*, vol. 84, no. 7, pp. 917–930, July 1996.

[3] V. Bhaskaran, K. Konstantinides, and B. R. Natarajan, "Multimedia architectures: From desktop to portable appliances," *Proc. SPIE*, vol. 3021, pp. 14–25, 1997.

[4] F. Chen, J. D. Villasenor, and D. S. Park, "A low-complexity rate-distortion model for motion estimation in H.263," in *Proc. 3rd IEEE Int. Conf. Image Processing*, vol. 2, Sept. 1996, pp. 517–520.

[5] B. Girod, E. Steinbach, and N. Farber, "Performance of the H.263 video compression standard," *J. VLSI Signal Proc. Syst. for Signal, Image, Video Technol.*, vol. 17, no. 2–3, pp. 101–111, Nov. 1997.

[6] Z. L. He and M. L. Liou, "A high performance fast search algorithm for block matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 826–828, Oct. 1997.

[7] D. Y. Hsiau and J. L. Wu, "Real-time PC-based software implementation of H.261 video codec," *IEEE Trans. Consumer Electron.*, vol. 43, pp. 1234–1244, Nov. 1997.

[8] *Generic Coding of Moving Pictures and Associated Audio*, ISO/IEC Draft International Standard 13 818-2, Nov. 1993.

[9] *Video codec for audiovisual services at $p \times 64$ kbits/s*, Recommendation H.261, 1990.

[10] *Video coding for low bit-rate communication*, ITU-T Draft Recommendation H.263, Dec. 1995.

[11] L. Kohn, G. Maturana, M. Tremblay, A. Prabhu, and G. Zyner, “The Visual Instruction Set (VIS) in UltraSPARC,” in *Proc. COMPCON*, Spring 1995, pp. 462–469.

[12] B. R. Lee, K. K. Park, and J. J. Hwang, “H.263-based SNR scalable video codec,” *IEEE Trans. Consumer Electron.*, vol. 43, pp. 614–622, Aug. 1997.

[13] B. Liu and A. Zaccarin, “New fast algorithms for the estimation of block motion vectors,” *IEEE Trans. on Circuits Syst. Video Technol.*, vol. 3, pp. 148–157, Apr. 1993.

[14] Z. J. A. Mou, D. S. Rice, and W. Ding, “VIS-based native video processing on UltraSPARC,” in *Proc. 3rd IEEE Int. Conf. Image Processing*, vol. 2, Sept. 1996, pp. 153–156.

[15] K. N. Ngan, D. Chai, and A. Millin, “Very low bit rate video coding using H.263 coder,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 308–312, June 1996.

[16] H. S. Oh and H. K. Lee, “Adaptive rate control scheme for very low bit rate video coding,” *IEEE Trans. Consumer Electron.*, vol. 42, pp. 974–980, Nov. 1996.

[17] A. Peleg, S. Wilkie, and U. Weiser, “Intel MMX for multimedia PCs,” *Commun. ACM*, vol. 40, no. 1, pp. 25–38, Jan. 1997.

[18] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications, Appendix A-2*. New York: Academic, 1990.

[19] K. Rijkse, “H.263: Video coding for low-bit-rate communication,” *IEEE Commun. Mag.*, vol. 34, no. 12, pp. 42–45, Dec. 1996.

[20] H. Sava, M. Fleury, A. C. Downton, and A. F. Clark, “A case study in pipeline processor farming: Parallelising the H.263 encoder,” in *Proc. BCS PPSG, U.K. Parallel 1996*. New York: Springer-Verlag, July 1996, pp. 196–205.

[21] E. Steinbach, N. Farber, and B. Girod, “Standard compatible extension of H.263 for robust video transmission in mobile environments,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 872–881, Dec. 1997.

[22] Sun Microsystems, *Visual Instruction Set (VIS) User’s Guide*, Mar. 1997, Version 1.1.

[23] Telenor Research and Development, *TMN (H.263) Coder, Version 2.0* Norway, 1996.

[24] M. Tremblay, D. Greenley, and K. Normoyle, “The design of the microarchitecture of UltraSPARC-I,” *Proc. IEEE*, vol. 83, pp. 1653–1663, Dec. 1995.

[25] C. G. Zhou *et al.*, “MPEG video decoding with the UltraSPARC visual instruction set,” in *Proc. COMPCON*, Spring 1995, pp. 470–475.

Shahriar M. Akramullah (S’96–M’99) was in Dhaka, Bangladesh. He received the B.Sc. degree in electrical engineering from Bangladesh Institute of Technology, Chittagong, in 1991, ranking second among a class of 1966. He received the M.Phil. and Ph.D. degrees in electrical engineering from the Hong Kong University of Science and Technology (HKUST), Kowloon, Hong Kong, in 1995 and 1999, respectively.

Currently, he is engaged in post-doctoral research in the Electrical and Electronic Engineering Department, HKUST. He has published several papers in distinguished international journals and conference proceedings. His research interests include video and multimedia signal processing, parallel processing, and distributed algorithms for real-time video and image processing.

Dr. Akramullah was a recipient of the Commonwealth Scholarship from 1993 to 1995 and the HKUST PG studentship from 1996 to 1999. He received the Best Paper Award and the Distinguished Paper Award in the IEEE (Hong Kong Section) PG Student Paper Contest in 1995 and 1997, respectively. His biographical profile is listed in the 2000 *Who’s Who in the World*. He is a member of ACM.

Ishfaq Ahmad received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985, the M.S. degree in computer engineering, and the Ph.D. degree in computer science, both from Syracuse University, Syracuse, NY, in 1987 and 1992, respectively.

He is an Associate Professor in the Department of Computer Science, Hong Kong University of Science and Technology (HKUST), Kowloon, Hong Kong. He is also the Director of HKUST’s Multimedia Technology Research Center, which is supported through research grants from government funding agencies and several Hong Kong and U.S. companies, and whose focus is on research and applications of interactive multimedia systems and video compression, in collaboration with some 50 industrial partners and research laboratories around the world. His research interests span the areas of high-performance computing, parallel and distributed systems, video compression, and networked multimedia information technologies. He has published over 100 technical papers in journals and conferences.

Dr. Ahmad serves on the Editorial Boards of *Journal of Parallel and Distributed Computing*, *IEEE Concurrency* (Cluster Computing), IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, and the IEEE Distributed Systems Online.

Ming L. Liou (F’79) received the B.S. degree from National Taiwan University, the M.S. degree from Drexel University, Philadelphia, PA, and the Ph.D. degree from Stanford University, Stanford, CA, in 1956, 1961, and 1964, respectively, all in electrical engineering.

He joined AT&T Bell Labs in 1963 as a member of Technical Staff and held various supervisory positions until 1984, when he joined Bellcore, and became the Director of the Video Signal Processing Research Group. He is currently with Hong Kong University of Science and Technology, Kowloon, Hong Kong, where he is Professor Emeritus. He has published numerous technical papers.

Dr. Liou has been active in professional activities and has served in various capacities, including as Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS from 1979 to 1981, President of the IEEE Circuits and Systems (CAS) Society in 1988, and Founding Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY from 1991 to 1995. He received the CAS Society Special Prize Paper Award in 1973 and the Darlington Prize Paper Award in 1977. He is a Fellow of HKIE and a member of Sigma Xi, Eta Kappa Nu, and Phi Tau Phi.