REAL-TIME H.264/AVC CODEC ON INTEL ARCHITECTURES

Vaughn Iverson, Jeff McVeigh, Bob Reese

Intel Corporation

ABSTRACT

The impressive coding efficiency of H.264/AVC comes at the expense of significantly increased algorithmic complexity compared to existing standards, which has limited the availability of cost-effective, high-performance solutions. We present a design overview and implementation details of a compliant codec optimized for the Intel® Pentium® 4, Pentium® M, and XScale™ micro-architectures, including results for encoder rate-distortion performance, and encoder/decoder execution speed on the above processors. These results demonstrate that high-quality, real-time H.264/AVC recording and playback are achievable on today's PC (desktop and notebook) platforms, and that real-time playback on handheld (mobile phone / portable media player) platforms can also be achieved, with both implemented as optimized software for these general-purpose processor architectures.

1. INTRODUCTION

The overarching design goal of the H.264 [1] / MPEG-4 Advanced Video Coding (AVC) [2] standard was to achieve improved coding efficiency compared to previous standards; with a target of requiring 50% lower bit rate at equivalent visual quality. While the actual compression efficiency depends directly on the encoder design and implementation, clear evidence exists that compliant encoders are capable of achieving this aggressive goal across low and high bit rate content [3].

The standard utilizes the familiar block-based motion-compensated, transform coding structure. It also employs a number of advanced coding tools, which collectively provide the desired coding efficiency improvements (see [3] for a complete overview of the coding algorithm). The use of these advanced coding tools comes at the expense of significantly increased computational complexity. Reported complexity analyses have shown that an H.264/AVC baseline decoder is approximately 2-2.5X more complex than a similarly implemented H.263 baseline decoder [4]. While encoder complexity is more difficult to quantify, we estimate that an H.264/AVC encoder can be upwards of 5-10X more complex than an H.263 encoder.

This additional complexity directly translates into increased implementation cost. For a hardware-based implementation (e.g. ASIC), increased logic gate count, clock frequency, memory bandwidth, and memory storage are required to achieve a desired level of encode/decode performance. Similarly, increased processor clock frequency, memory bandwidth, and memory storage are required for a software-based implementation on a programmable platform. Since these advanced tools differ considerably from those in other video coding algorithms, it is difficult to reuse components from previously developed implementations, often resulting in a complete system redesign. These factors have limited the availability of cost-effective, high-performance solutions.

To enable H.264/AVC operation on current and future Intel platforms and to speed adoption of the standard, we have developed a Baseline Profile encoder/decoder optimized for the Intel Pentium 4, Pentium M and XScale micro-architectures. Our encoding algorithm yields compression efficiency performance within approximately 20% of the latest reference encoder, while operating roughly 200 times faster, with real-time operation possible on today's Pentium 4 and Pentium M platforms at Level 3. Similarly, our optimized decoder is capable of real-time operation at Level 3.2 on Pentium 4, Level 3.1 on Pentium M, and Level 2.2 on currently shipping XScale platforms.

The remainder of the paper is structured as follows. Section 2 provides an overview of the target Intel architectures. The design of the encoder / decoder is provided in Section 3. Performance results and analysis are detailed in Section 4, and we provide conclusions in Section 5.

2. OVERVIEW OF INTEL ARCHITECTURES

Our codec development was focused on Intel processors that target the mainstream consumer market segment. These include the Pentium 4 processor for desktop PCs, the Pentium M processor for notebook PCs, and the XScale processor for cell phones, PDAs, and portable media players.

The most recent processors in the Pentium 4 family, (formerly codenamed Prescott), are built on new 90 nm process technology and incorporate an enhanced Intel NetBurst® architecture, deeper pipelining for higher frequency operation, a larger Level 2 (L2) cache, and new SSE3 Instructions [5]. SSE3 is the latest generation of Single Instruction Multiple Data (SIMD) support, following the 64-bit SIMD Intel® Wireless MMX™ technology, and 128-bit SIMD Intel SSE and SSE2 technologies, which are useful for high performance video codec development. The new 1MB on-chip L2 cache further improves the performance of memory-intensive video processing, especially high definition content.

The Pentium M processor utilizes the same instruction set architecture as the new 90 nm Pentium 4 processor (except for SSE3 support), and has a similar 1 MB L2 cache. It also includes a power-optimized system bus, micro-ops fusion and a dedicated

0-7803-8554-3/04/$20.00 ©2004 IEEE.

stack manager for fast instruction execution while consuming less power [6].

The next-generation Intel XScale processor, the PXA270, is based on the ARMv4 instruction set architecture [7]. The PXA270 processor also includes Intel® Wireless MMX™ technology (WMMX), which combines MMX instructions, integer instructions from SSE2, and several brand new multimedia acceleration instructions to enable 64-bit SIMD processing [8].

Processor	Pentium 4	Pentium M	PXA270
Avail. frequency	3.4 GHz	1.7 GHz	624 MHz
SIMD support	MMX, SSE, SSE2, SSE3	MMX, SSE, SSE2	WMMX
L1 cache (Data, Instruction)	16 KB (D+I)	32 KB (D), 32 KB (I)	32 KB (D), 32 KB (I)
L2 cache	1 MB (D+I)	1 MB (D+I)	None
System bus	800 MHz	400 MHz	208 MHz

Table. 1. Architecture summary.

3. CODEC DESIGN

The codec was designed with the following overall goals: fast execution, portability across processors and operating systems, and maintainability for future research and development. Our current implementation is fully Baseline Profile compliant and additionally supports B slices and CABAC (decode only). It is thoroughly optimized for Pentium 4, Pentium M, and XScale processors.

The conflicting design goals of high performance and portability / maintainability were addressed through the use of C++ code for the bulk of the codec, with processor-specific assembly code-paths for computationally intensive functions. The assembly-coded functions are called through function pointers initialized at build or run time.

3.1. Encoder Design

A real-time encoder must balance the objectives of achieving the lowest compressed bit rate for a desired level of subjective quality while satisfying real-time constraints for the target platform. Since the standard only defines the decoding process, there exists considerable flexibility in the choice of encoding algorithms to meet these objectives.

To balance bit rate and quality, our encoder employs a common rate-distortion (R-D) optimization process for virtually all coding decisions, such as motion vector and mode selection [9]. For potential block coding decisions, the measured distortion between the original and reconstructed pictures for a given choice is adjusted by a R-D factor estimated from the number of bits to code side-information for that choice. The choice with the lowest rate adjusted distortion is selected. The number of bits is typically calculated directly from the known cost of coding fields in various bitstream headers. The R-D factors are calculated from look-up tables that are empirically tuned for each type of decision at each quantization level. This process minimizes situations where a particular mode may yield a very low distortion (good visual quality), but requires a disproportionate number of bits to code the associated side-information.

To achieve real-time operation, our encoder utilizes a combination of well-known and novel algorithmic techniques to dramatically reduce the computational complexity compared to the reference encoder, with only minor degradation in rate-distortion performance. These simplifications are mainly targeted at the motion estimation process, as it is the most quality and performance critical section. The encoder also supports an adjustable complexity setting to enable more exhaustive encoding algorithms on higher performance platforms.

Our first simplification was to omit the use of multiple, long-term reference frames in the encoder due to the significant computational complexity and relatively limited coding gain for general-purpose content.

The following process is a high-level overview of the mode selection and motion estimation techniques used for our "medium complexity" encoder setting. Steps 1 & 2 are performed for the whole frame in advance; all others are done per 16x16 macroblock (MB) in succession. Steps 2, 4 & 5 are done for each reference frame and Steps 2-7 are omitted for I slices.

Step 1 – Edge detection: A low-complexity edge detection algorithm determines whether an edge is present in each MB. This result is used at the end of the coding process for inter vs. intra coding selection (see Step 8).

Step 2 – "Signature" motion estimation search: This coarse-level algorithm searches using a "signature" representation of each MB, reducing the distortion calculation cost by a factor of four horizontally and vertically. The result is an approximate motion vector for each MB.

Step 3 – Empty/Direct "early-out" decision: An R-D optimized distortion calculation is performed for use of an Empty (P slice) or Direct mode (B slice) MB. If the result is below a QP dependent threshold, additional motion estimation is skipped (go to Step 8).

Step 4 – Calculate starting motion vector: The best motion vector for each MB is selected from six vector candidates: the zero vector, predicted vector, above vector, left vector, the vector from the upper left block of the co-located MB in the reference frame (scaled appropriately), and the vector from Step 2.

Step 5 – Integer refinement and block statistics: An exhaustive integer refinement (±4 samples in each dimension) is performed using the starting vector from Step 4. Statistics are calculated for each of the 41 possible sub-blocks, with distortions calculated at the level of 4x4 blocks. Distortions for blocks larger than 4x4 are formed by summing together the appropriate block statistics.

Step 6 – Block splitting and reference frame selection: Based upon the 41 (82 for B slices) motion vector / distortion pairs, R-D optimized MB and block splitting, and reference frame selection is performed. For B slices, Bi-Prediction for each sub-block and 8x8 and 16x16 direct modes are also considered..

Step 7 – Fractional sample motion vector refinement: All motion vectors are refined to ¼ sample resolution using a 4-step conjugate search that considers a total of 8 points by successively considering ½ and then ¼ sample resolutions horizontally and vertically, selecting the best vector from the three alternatives at each step on a rate-distortion optimized basis.

Step 8 – Intra mode selection and final mode decision: If the distortion from Step 7 above is below a certain threshold, this step is skipped; otherwise a mode and distortion are calculated for the 16x16 intra mode. If the result of Step 1 indicated an

edge in this MB, then the best 4x4 advanced intra prediction modes are considered as well, using the original (unreconstructed) picture data for predictors. The overall best intra mode is then compared to the results of Step 7, yielding the final intra or inter-coding mode decision.

Motion compensation, transform, quantization, reconstruction and variable length coding are then performed, in many cases reusing saved predictors calculated during the above analysis. PCM mode is considered only after coded bit-count for a MB is complete. If the macroblocks in a P or B slice have predominantly been coded using intra modes, the entire slice is recoded as an I slice. Finally the H.264/AVC loop-filter is applied to the entire frame.

3.2. Decoder Design

High performance decoding is achieved through selective use of assembly code, and by minimizing data sets and maximizing cache locality to minimize wasted memory stall cycles by sizing buffers to the smallest data type instead of the natural 32-bit type.

Use of assembly code to take advantage of processor-specific instructions and capabilities is an important factor in maximizing performance. Examples in the decoder are SIMD (SSE3/SSE2/WMMX) for fractional-sample motion compensation filters, the deblocking filter, and the inverse transform and add functions.

The deblocking filter does not readily translate to a SIMD implementation due to the many conditional filter choices – at the macroblock level to select the filter strength for each 4-sample block edge (including possibly no filter) – and at the individual sample level where filter thresholds (again possibly not filtering) are dependent upon adjacent sample values.

For example, the SSE3 128-bit instructions can be used to filter 8 samples (two block edges) in parallel. Each of the two edges can be filtered or not, and on a filtered edge, each of the 4 samples can be filtered or not, depending upon adjacent sample values (there are additional filter details, omitted for clarity).

The SIMD solution uses non-conditional instructions to perform filter operations for all 8 samples, computing multiple results, encompassing all possible outputs, for each sample. Data-dependent masks are then used with AND and OR operations to choose the final result with no branching. This approach exploits SIMD parallelism while avoiding the performance penalties associated with the many data-dependent mispredicted branches in a non-SIMD implementation.

To further achieve data cache locality the decoder is divided into 3 phases for each frame (illustrated in Fig 1). Each phase is run independently for all MBs of the frame. Because a frame decode typically processes MBs in raster order, in each of these phases the data referenced when processing a MB is likely to be resident in the data cache from processing the previous MB.

An alternate approach of executing multiple phases for each MB results in more cache contention and worse performance. The phased approach has an added cost of additional buffering of data from one phase to the next, but experimentation has demonstrated the choice to be a good performance tradeoff. The largest such buffer is for block coefficients, minimized by storing only quantized non-zero coefficients and a block position instead of all 16 coefficients for a 4x4 block.

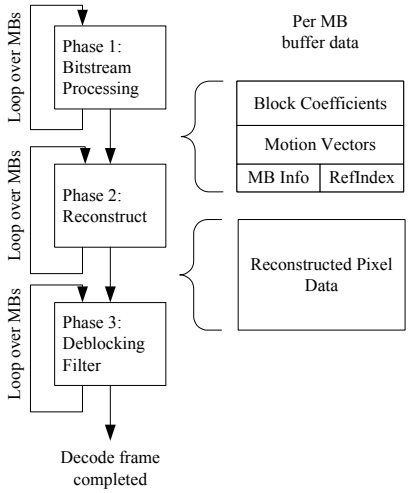

Fig. 1. Decoding process flow and intermediate phase buffer storage requirements.

4. RESULTS AND ANALYSIS

All sequences were encoded with our encoder using Baseline Profile tools, one reference frame, a motion vector search range of ±15 samples, one slice per frame, and fixed quantization. All software on each platform was compiled using full optimization.

Processor	Pentium 4	Pentium M	PXA270
Frequency	2.8E† GHz	1.6 GHz	403 MHz
RAM	1 GB	1 GB	64 KB
Operating Sys	Windows XP	Windows XP	WinCE 4.2

Table. 2. System configurations.

We have performed extensive analysis across a wide-range of sequences and bit rates. Due to space limitations, we only report results for representative sequences here. Figure 2 provides a rate-distortion comparison between our encoder and the JM8.0 reference encoder for the Foreman sequence. On average, the reference encoder is able to achieve equivalent PSNR while consuming approximately 20% lower bit rate. This reduced rate-distortion performance for our encoder is due to the various algorithmic simplifications employed to achieve real-time performance. Specifically, our encoder operates at 83 frames/second (fps) for the Foreman sequence (CIF, 586 Kbps, see Fig. 2) and 24 fps for the Stefan sequence (720x480, 2 Mbps) on the Pentium 4 platform. This is approximately 300 times faster than the reference encoder. Encoding on the Pentium M platform is approximately 25% slower than on the Pentium 4 platform. The PXA270 platform encodes at ~7 fps for the Foreman sequence.

Decoding performance results across bit rates are provided in Figs. 3 and 4, where the IP/IQ/IT/ADD value is the time spent in intra prediction, inverse quantization, inverse transform, and block addition. Real-time decode of the Foreman sequence (CIF, 30 frames/second, 200-800 Kbps) consumes only 5-9% of the

† The 2.8E GHz Pentium 4 processor includes a 1MB L2 cache, 800 MHz system bus and SSE3 instructions.

Pentium 4 platform, 5-11% of the Pentium M platform, and 47-67% of the PXA270 platform.

Decoding of high-definition content (Cyclists sequence, 1280x720, 4 Mbps) operates at 65 fps on the Pentium 4 and 46 fps on the Pentium M platform.

To examine the effect of multiple reference frames on decoder performance, we used the reference encoder to generate sequences using five reference frames. The optimized decoder ran approximately 20% slower on Pentium 4 and Pentium M processors and 10% slower on the PXA270 processor compared to sequences with one reference frame.

In summary, our Baseline Profile encoder on Pentium 4 and Pentium M processors easily handles real-time operation for Level 2.1 content (up to 352x480, 30 fps), and is capable of real-time encoding for some Level 3 content (720x480, 30 fps). Our real-time decoder can support Level 3.2 sequences (1280x720, 60 fps) on Pentium 4 'E' systems with SSE3 instructions, Level 3.1 sequences (1280x720, 30 fps) on Pentium M systems, and Level 2 sequences (352x288, 30 fps) on PXA270 systems.

5. CONCLUSION

We presented the design and implementation details of a compliant H.264/AVC codec optimized for Intel architectures. The rate-distortion and execution speed results demonstrate that high-quality, real-time recording and playback is achievable on today's PC and handheld platforms. A sample version of the presented decoder is available for use at [10].

6. REFERENCES

[1] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services," May 2003.

[2] ISO/IEC 14496-10:2003, "Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding", October 2003.

[3] T. Wiegand, G. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560-576, July 2003.

[4] M. Horowitz, A. Joch, F. Kossentini, and A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 704-716, July 2003.

[5] Intel Corp., "Prescott New Instructions Software Developer's Guide," Order no. 252490-003, June 2003.

[6] Intel Corp., "Intel® Pentium® M Datasheet," Order no. 252612-002, June 2003.

[7] "Intel Unveils New Technologies for Future Cell Phone, PDA Processors Based on Intel® XScale® Technology", www.intel.com, September 2003.

[8] "Intel® Wireless MMX™ Technology", www.intel.com.

[9] G. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," IEEE Signal Proc. Magazine, vol. 15, pp. 74-90, Nov. 1998.

[10] "Intel Integrated Performance Primitives Version 4.0," www.intel.com.

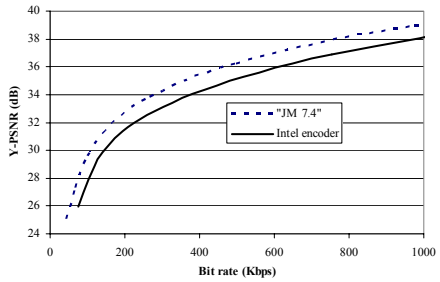

Fig. 2. Encoder rate-distortion for Foreman (CIF, 30 fps).

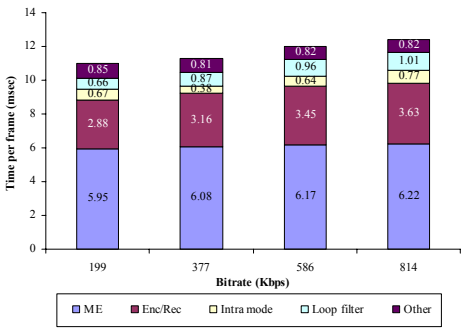

Fig. 3. Average encoding time per frame for Foreman (CIF, 30 fps) on the 2.8E GHz Pentium 4 platform.

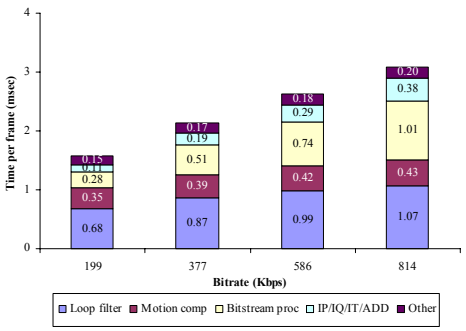

Fig. 4. Average decoding time per frame for Foreman (CIF, 30 fps) on the 2.8E GHz Pentium 4 platform..

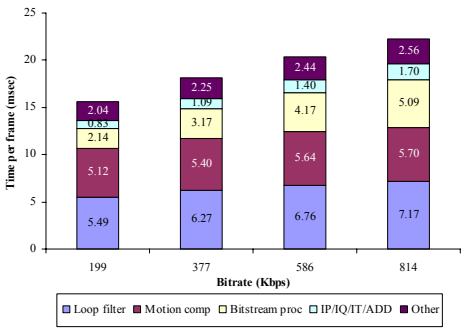

Fig. 5. Average decoding time per frame for Foreman (CIF, 30 fps) on the XScale PXA270 platform at 403 MHz.