Algorithms and DSP Implementation of H.264/AVC

Hung-Chih Lin, Yu-Jen Wang, Kai-Ting Cheng, Shang-Yu Yeh, Wei-Nien Chen, Chia-Yang Tsai, Tian-Sheuan Chang, Hsueh-Ming Hang
Dept. Electronics Engineering, and Institute of Electronics, Hsinchu 300, Taiwan
e-mail: hclin.ee93g@nctu.edu.tw, cosbe@twins.ee.nctu.edu.tw, kt34.ece90@nctu.edu.tw, chucky1984820.ee91@nctu.edu.tw, tpht78@hotmail.com, cytsai.ee90g@nctu.edu.tw, tschang@twins.ee.nctu.edu.tw, hmhang@mail.nctu.edu.tw

Abstract - This survey paper intends to provide a comprehensive coverage of the techniques that are pertinent to the processor-based implementation of H.264/AVC video codec, particularly on DSP. Most of this paper is devoted to the computationally efficient algorithms, or the *fast algorithms*. Fast algorithms for motion estimation, intra-prediction and mode decision are described to reduce the computational complexity. In addition, in order to port the H.264/AVC codec to DSP, we also outline the basic principles of DSP code optimization.

I. Introduction

ITU H.264 Advance Video Coding (AVC), also known as the MPEG-4 part 10 [1], offers the highest coding efficiency among all the existing video compression standards for, particularly, very low rate video transmission. However, it also has the highest computational complexity. Therefore, reducing its implementation complexity becomes a very challenging subject.

Numerous studies on reducing H.264/AVC codec implementation complexity have been published in the past 3 years since this standard was finalized in late 2002. The purpose of this survey is to give a comprehensive treatment of the techniques that are pertinent to the processor-based implementation of H.264 codec. Although H.264 is a very new standard, its literature is abundant. Limited by space and our knowledge, we will describe the approaches that, based on our experiences, have good potential in constructing a DSP-based codec. In general, the encoder part can be speedup by various fast algorithms to save the computation, while both encoder and decoder can be accelerated by the processor-dedicated parallel processing instructions.

The rest of the paper is organized as follows. In Section II, we first brief review the H.264 video standard and its computational profile. Section III contains a short discussion on the general principles of accelerating an algorithm implemented on a processor. Then, we present the fast algorithms for the intra prediction, motion estimation, mode decision and other parts in Section II to Section VI, respectively. Then we show the code speed-up tips for DSP in Section VII. Finally, a few conclusion remarks are made in Section VIII.

II. Overview of H.264 Video Coding

A. Overview

H.264 consists of a number of tools. Its basic structure is the so-called motion-compensated transform coder. Compared to the prior video coding standards, many important and new techniques are employed in H.264 and they together bring significant improvement on coding performance. Some of these techniques are highlighted here [2]. We may want to add that the concepts of some of these tools have existed for some time but they are nicely tuned and integrated together to form a good compression scheme in H.264.

1) Variable block-size motion compensation with multiple references

The basic unit in H.264 motion estimation is the 16x16 macroblock. It can be further split into a tree structure, with a minimum motion compensation block size as small as 4x4. Also, up to five reference frames may be used for motion compensation.

2) Directional spatial intra coding

To reduce the correlation inside a block, H.264 adopts the intra-prediction technique, which estimates the current block pixel values based on the known pixels of its neighbor blocks. The prediction results implicitly follow the edge direction, and often bring significant improvements.

3) In-loop deblocking filter

Block-based video coding produces artifacts known as blocking artifacts at low bit rates. This in-loop deblocking filter adjusts its filter strength adaptively according to the image local characteristics, and thus it provides better quality pictures at the decode end.

4) Context adaptive entropy coding

Two entropy coding methods, Context-based Adaptive Binary Arithmetic Coding (CABAC) and Context-based Adaptive Variable Length Coding (CAVLC), are provided in H.264. Both methods use context-base adaptivity to improve the entropy coding performance and the results show this approach is quite successful.

A simplified encoding flow of H.264 is shown in Fig. 1. A video frame is first partitioned into a number of 16x16 macroblocks. Then, each macroblock goes through the intra-prediction or the inter-prediction unit. The intra prediction unit uses the neighboring block data to predict the current block. The inter-prediction uses reference frames to predict the current frame. Each predictor has a number of modes. A good design should pick up the best mode with the lowest rate and distortion. The prediction residuals are then transformed, quantized and further entropy-coded into the output bitstream. In order to continue operating on the next incoming frame, the quantized current frame is reconstructed and stored. The decoder data flow is the reverse of the encoder flow.

B. Computational profile

The H.264 encoder reference software provided by the ITU/MPEG standard committee is known for its high computational complexity. A typical computational profile of the

H.264 encoder (ITU/MPEG reference software) running on Intel PC, is shown in Fig. 2. It shows that the tools of (a) motion estimation, (b) entropy coding, (c) transform and quantization, (d) interpolation, and (e) mode decision and intra-prediction are the most time-consuming modules. Although the other processors would have somewhat different architectures from the Intel processor, by and large, the trend is pretty much the same. As for the decoder, the tools of (a) motion compensation (including interpolation), (b) entropy decoding, and (c) intra-prediction have the CPU load.

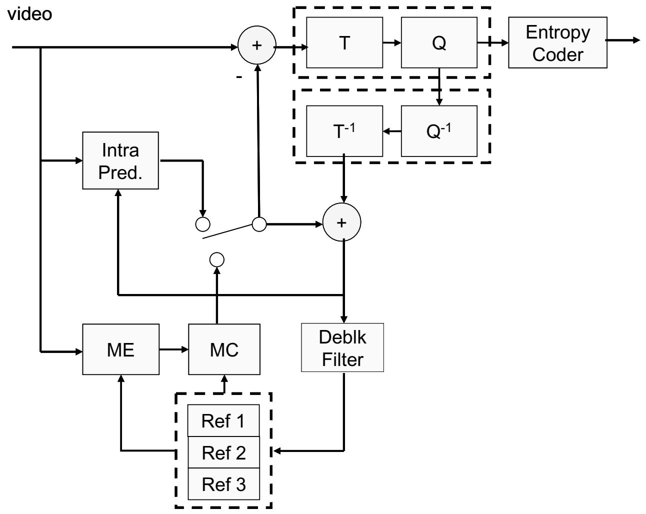

Fig. 1. Block diagram of H.264 encoder

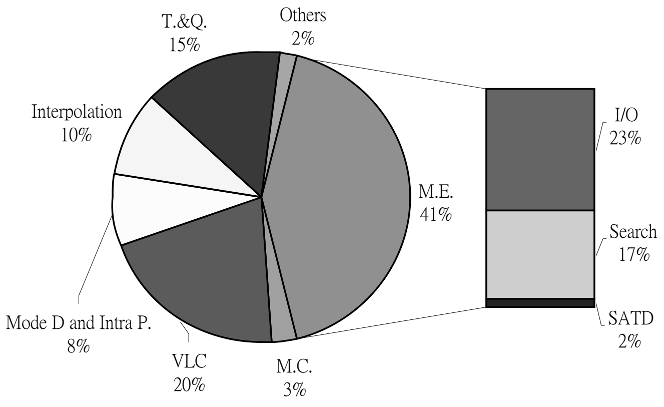

Fig. 2 Computational profile of H.264 video encoding.

III. Acceleration Methodology

The focus of this paper is efficient implementations of H.264 on a DSP system. Limited by the computing power and memory size of DSP, we need to modify the original software to reduce its computational complexity and to match the DSP computing architecture.

Essentially there are two types of calculation acceleration steps. The first type is programming techniques that reduce the redundancy in the execution codes and alter the program to match the DSP structure, for example, loop unrolling. The second type is to replace certain complicated modules by their approximation counter parts. That is, in contrast to the first type of speed-up process that does not change the output values, the second type speed-up process changes the output values. Our target is to find the fast algorithm modules that approximate the original modules well and, therefore, little performance degradation is encountered.

Typically, we first analyze the current program complexity by profiling its execution as we did in the last section. After identifying the most computational intensive modules, we look for proper acceleration steps. In the case of H.264, the decoder is rigidly specified by the standard and thus generally only the first type acceleration steps can be used so that the output values are precisely reserved after acceleration. On the other hand, the encoder is not completely specified by the standard and thus there is quite a lot of flexibility in the encoder. We thus look for good fast algorithms that replace the original modules without sacrificing the compression efficiency.

IV. Fast Algorithms for Intra Prediction

A. Overview

Intra-prediction uses the high correlation property of neighboring samples in spatial domain to predict the current encoded samples. For the luma samples, each prediction block may be formed for each 4x4 block (denoted as I4MB) or for an entire MB (denoted as I16MB). When utilizing Intra_4x4 prediction, each 4x4 block chooses one of the nine prediction modes, which include one DC mode plus eight directional prediction modes, as shown in Fig. 3 (left), as the best one. In the luma component of an MB, the Intra_16x16 prediction is typically chosen for smooth image areas, and thus, only four prediction modes are specified as shown in Fig.3 (right) except for the DC mode. The chroma samples of an MB are predicted using a similar prediction pattern, Intra_8x8, which is similar to the luma Intra_16x16 prediction.

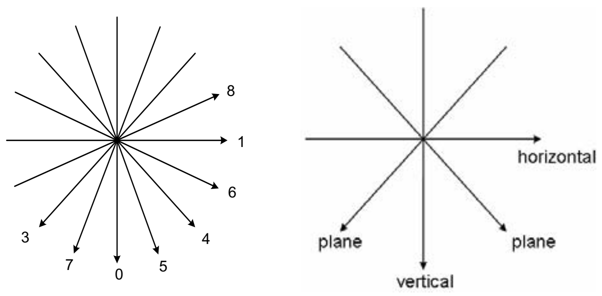

Fig. 3 Intra prediction modes for Intra_4x4 (left) and Intra_16x16 (right).

B. Fast algorithms

The fast algorithms of intra prediction can be classified into several types. The first approach is "early termination", which ends the search operation when the calculated distortion is smaller than a pre-chosen threshold. The selection of a proper measure for deciding termination is critical to the performance. It may be derived based on the macroblock smoothness [3][4] or the most probable mode [5]. The early termination based on the macroblock smoothness calculates a smoothness measure of a macroblock to determine the block type. For example, the large block type such as Intra_16x16 is chosen often for the flat image areas [3][4]. "Smooth" means that all the pixel values in a MB are similar; that is, their variance is small. The variance computation shall be simple to save computation. Therefore, the Mean-Absolute-Difference (MAD) operation [3] or the AC/DC ratio [4] is often used. If the variable is smaller than a pre-selected threshold value, the Intra_16x16 mode is chosen and thus the costly Intra_4x4 can be skipped.

Another kind of early termination proposal examines the most probable mode first. For example, in searching for the best Intra_4x4 mode, if its residual is smaller than a threshold, then the other eight Intra_4x4 modes are skipped (not chosen). Otherwise, all nine modes have to be tested. Then, we set another threshold to decide whether to keep on checking the Intra_16x16 prediction or not. It was reported that in one case, this method together with the 2:1 downsampling and rate-distortion optimization (RDO) can reduce 68.8% of total computation time with only 1.35% of bit rate increase comparing to the reference software [5]. The major issue in this type of algorithms is how to determine the threshold. The threshold value can be adjusted according to the quantization parameters for instance. To construct a more efficient scheme, we propose a mixed fast intra prediction algorithm. It first examines both the most probable mode and the DC mode to determine if it meets the early termination criterion. The threshold value is decided by the average of SATD (sum of absolute transformed difference) of all the previous Intra_4x4 blocks in this frame. Once the 16 Intra_4x4 blocks are done, their total cost will be used as the threshold for deciding Intra_16x16 mode. These threshold values seem to be able to match the video local characteristics and provide good results. Even when RDO is turned off, we can achieve around 30% computational savings for the intra prediction module.

The second approach uses the edge analysis to quickly identify the edge direction since the intra prediction is basically a directional prediction [6][7]. Often the Sobel operators or the first order derivative are used as the edge analysis tool to find the most probable edge, which will be used as one of the final edge candidates. The final mode candidate list includes the one selected by the edge detector together with the other highly probable modes. In the case Intra_4x4, this would mean two modes of the neighboring blocks and the DC mode; and in the Intra_16x16 and Intra_8x8 cases, only the DC mode is considered highly probable. Therefore, only four candidate modes (for Intra_4x4) or two candidate modes (other types) are needed to be examined. The result shows that 60% of intra_only computation time reduction is observed with RDO and the bit rate increase is around 2~3% [6]. The bit rate increase may be owing to the irregular edges within a block. On the other side, the extra computation needed for edge analysis can be a computation burden and reduce the overall saving significantly.

The third approach uses the so-called three step approach [8]. It first tests the horizontal and vertical directions, it then tests the neighboring 22.5 degree modes close to the better one from the previous step, and finally the best mode up-to-now is checked against the DC mode for the final winner. This approach has the advantage of a fixed number of modes are examined for all cases. However, computation time reduction is around 33% with about 1% bit rate increase.

The last approach makes use of the correlation in the temporal domain [9] since the best prediction mode in the current macroblock is likely similar to that in the reference macroblock in the previously coded frame(s). Thus, the primary intra prediction mode is selected from the mode of the most overlapped block in motion estimation. The computational overhead is nearly zero since all information is obtained during the inter-prediction operation. It is reported that the coding performance is nearly unchanged while the computational savings is about 50% assuming the intra-frame period is 10 [9].

In summarizing various fast intra-prediction algorithms, although we cite the experimental results from the proposed documents, a fair comparison among all methods is difficult because their simulation environments are quite different. One important element affecting computation is the option of RDO in the reference software. This is particularly true for the early termination method with thresholds.

The algorithms described in the above can be combined together to achieve further speed-up. For example, the first step could be the decision on Intra_4x4 or Intra_16x16. The second step could be the early termination for the chosen intra type. Finally, the rest of mode tests could be a fast algorithm to select one from the nine or four candidate modes.

V. Fast Algorithms for Motion Estimation

A. Overview

Block matching based motion estimation and compensation is a fundamental process in the current international video compression standards. It can efficiently remove interframe redundancy. A direct implementation is the full search algorithm that examines exhaustively every candidate motion vector in the search window to find the globally best matched block in the reference frame. However, its computationally intensive nature prevents it from practical implementation on a processor for real-time applications. The computation burden is increased drastically for the H.264 encoder because there are a number of combinations of partitioning a macroblock into sub-block(s) ranging from 4x4 to 16x16. Potentially each sub-block can have its own motion vector. This feature significant increases the computational complexity in motion estimation. Thus, many fast motion estimation algorithms have been proposed to alleviate the computational load.

Most of the fast algorithms are based on the well-known a priori knowledge, "the motion field of a real world image sequence is usually gentle, smooth and varies slowly". Fast motion estimation algorithms can be categorized into roughly three families as described below.

B. *Fast algorithms for motion estimation*

1) Searching over a subset of possible candidate points with certain search patterns

Based on the assumption of convexity of the unimodal error surface, i.e., block matching distortion increases monotonically away from the global minimum point, many gradient-based search methods with carefully designed search patterns have been developed to limit search points to a small subset of all possible candidates. This category includes the well-known three-step search (3SS) [10], the new three-step search (N3SS) [11], the cross search (CS) [12], the one-dimensional gradient descent search (1DGDS) [13], the block-based gradient descent search (BBGDS) [14], the four-step search (4SS) [15], the diamond search (DS) [16], the cross-diamond search (CDS)[17] and the hexagon-based search (HEXBS) [18]. Although this category of algorithms may be trapped into a local minimum point and hence the efficiency of the motion compensation may drop, they can considerably reduce the number of block matching computations.

2) Prediction of the motion vector based on correlations among motion vectors

Motion in most natural image sequences involves a few blocks and lasts for a few frames. Therefore, spatially or temporally adjacent blocks often have similar motion vectors. Taking the advantage of the correlation among neighboring motion vectors, the search window can be constrained to a small clique surrounding the “predicted vector”, a candidate position predicated based on the known neighboring motion vectors. Many prediction algorithms have been developed with different complexities. The prediction search algorithm (PSA) [19] simply predicts the current block motion vector as the mean value of its neighboring blocks’ motion vectors. Fuzzy search [20] applies fuzzy logic to predict the motion vector. In [21], motion vectors are predicted by integral projections. In [22], a spatial-temporal AR model of motion vectors is constructed and an adaptive Kalman filter is employed. The multi-resolution search [23] down-samples a picture to obtain raw motion vectors at different resolution levels, then it estimates finer motion vectors from the coarser ones. The multiresolution-spatiotemporal (MRST) scheme [23] modifies the normal raster scan order so that some blocks can reference more motion information by increasing their neighboring blocks along more directions. It then combines a multiresolution scheme and spatiotemporal correlation to predict motion vectors. For burst motions and blocks at the top-left corner, which has little correlation information, the performance of this category of algorithms may deteriorate because the refinement of prediction is restricted to a small search region. Moreover, the prediction overhead may reduce the speed gain.

3) Low complexity block matching criteria

The majority of the computations in motion estimation originate from computations of block matching distortion. In general, block matching metrics, such as the mean absolute difference (MAD) and the mean square error (MSE), involve pixel-wise operations, which are highly computationally intensive. Some methods try to simplify distortion computation by substituting the distortion defined on a subset of pixels for the whole block distortion. For instance, the MAD of 128 pixels is used as the matching distortion for a 16x16 macroblock in [23]; the computations can be reduced by one half with little performance loss. However, this method is not suitable for small blocks such as 4x4 blocks. Partial distortion elimination (PDE) in [24] compares every line’s distortion in a block to avoid computing the distortion of the entire block. In [25], hypothesis testing is used to estimate the MAD from the partial mean absolute difference (PMAD), and the estimated MAD value is used to judge the matching result.

When fast algorithms in the above three categories are put together, the motion estimation accuracy may degrade. Additional calculations such as the initial motion vector prediction could lead to a considerable amount of computational overhead.

An approach proposed without quality degradation is the successive elimination algorithm (SEA) suggested by Li and Salari [26], which pre-excludes some impossible candidate points before completing the matching distortion calculation. SEA is a fast full search algorithm having a performance identical to FS while it speeds up the search process approximately by 10 times for 16x16 macroblock based motion estimation. Some further improvements have been made in subsequent research [24][27]-[30].

4) Fast fractional motion estimation

In the H.264 video coding scheme [1], the inter prediction (motion vectors) precision has been increased to quarter pixel. Typically, people perform the integer pixel motion estimation (IME) first. Then, the sub-pixel motion estimation or fractional motion estimation (FME) is applied to achieve refinement. As compared to the integer-value search, FME has a somewhat different statistical character. This may due to the facts that the search window of FME refinement is much smaller than that of IME and that the referenced sub-pixels are interpolated from the integer-coordinate pixels. Consequently, the error surface of FME is much closer to a uni-modal one, which favors fast algorithms.

Therefore, traditional fast algorithms in IME can also be used and can be more effective. The scheme adopted by the H.264 reference software is a three-step-like fast algorithm. It first checks the nine candidates surrounding the best match of IME, and then checks further the nine candidates surrounding the best match from the previous step. However, to take even more advantage of the uni-modal surface property and the highly centralized distribution of sub-pixel motion vectors, several fast FME algorithms with additional features are proposed. In [31], a gradient based search algorithm is brought up. The search direction is determined first and looks for the best motion vector along that direction. In [32], an adaptive search-pattern algorithm is proposed. The search-pattern is determined by outcome of the previous step and it biased towards the search center. This method saves half of the computations when compared to the reference software.

5) Some recent approaches

The recent trend to further reduce the motion estimation calculations is to combine the techniques mentioned before. The idea is each technique, a fast algorithm, is placed its most suitable target area. Thus, how to find a specific combination that achieves the optimal solution for a specific application becomes the most important issue. In [33], a fast algorithm with better coding efficiency on residuals is proposed, which leads to a lower bit rate compared to the full search algorithm. The method proposed in [34] produces larger residuals (due to fewer search points) but less motion information. Overall, it has a better encoding efficiency and a rather fast coding speed. This type of solutions seems to the target now researchers are aiming at.

VI. Fast Mode Decision Algorithms

A. Overview

The mode decision algorithm determines the best mode of the macroblock from various combinations of inter-prediction and intra-prediction. It can be coded with seven different block sizes for motion-compensation in the inter mode, and various spatial directional prediction modes in the intra mode. To achieve the highest coding efficient as close as possible, the reference software calculates the rate distortion costs of all possible modes and the it chooses the best one that has the minimum cost. This is a very time-consuming process. To reduce the computation load, a fast mode decision algorithm is necessary, which can do a quick screening to drop most poor modes and then it examines the reminders and identifies the (nearly) best one.

B. Fast mode decision algorithm

The fast mode decision algorithm can be divided into two types. The first type uses an early termination threshold to terminate the lengthy mode decision process. The early termination step can be placed between the intra and inter prediction processes [35][36] or inside the inter prediction process [37].

The scheme proposed in [35][36] uses the fact that intra mode needs more bits for coding and thus has a lower priority than the inter mode. Thus, if the best inter mode cost is smaller than a threshold, the intra prediction mode is skipped. The threshold can be the average of rate distortion cost of a number of previously coded intra blocks [35] or a ratio between the average boundary error (ABE) and average rate (AR) [36], where AR is the average bits for encoding the motion-compensated residuals and ABE is the average pixel error between the pixels at boundary of the current and its adjacent blocks in the best inter mode. The simulation results show that it can achieve about 20% reduction of computational time with a slight bitrate increase.

In [37], it observes the fact that the 16x16 block usually is the best block size for large areas of background with still or uniform motion since it has less motion vector overhead. Thus, it first checks the cost of 16x16 block size. If it is smaller than a threshold, say, an average value of previous 16x16 blocks, the inter prediction process is terminated. Otherwise, a similar procedure is applied to the 8x8 block size.

The second type of the mode decision algorithms is to reduce the number of candidate modes. Intuitively, if the cost of a larger block-size mode is higher than the cost of the current block-size mode, the even larger block-size modes can be excluded. Similarly, if the cost of a smaller block-size is higher than that of the current block-size mode, the even smaller block-size modes can be excluded. Following this argument, we give different priority to each mode. If the mode with higher priority can provide sufficient image quality, we can skip the other lower priority modes. A specific case is the SKIP mode. The SKIP mode refers to the 16x16 mode of which no motion and residual information is coded. Thus, no motion search is required and it has the lowest complexity. Therefore, many algorithms assign the highest priority to the SKIP mode and thus a large percentage of macroblocks would get the SKIP mode based on spatial-temporal neighborhood information [38]-[40]. This approach can save a significant proportion of the encoding time with a slightly bit rate increase and quality drop.

In summary, the fast mode decision algorithms can be combined with the other fast intra and inter prediction algorithms to achieve further speedup. In all these algorithms, the SKIP mode first approach can save significant computational time. How to determine proper threshold values in a simple and automatic way is one critical issue for research and many proposals have been suggested.

VII. DSP Optimization for Video Codec

A. Overview

DSP processors made by different manufacturers vary in their functionality and capability. For real-time video codec implementation, the DSP processor shall have the parallel processing units and a wide data bus bandwidth to support the huge computational and memory access requirements. Almost all the high-end DSPs offered by several well-known vendors can meet these requirements. To make the following discussions more concrete, the popular TI's DSP is chosen as an example in this paper.

Tuning the video codec software for DSP implementation involves several steps. Traditional development flows in the DSP industry includes the following. Construct a C model for validating purpose. The C model is first run on a host PC or a UNIX workstation. Then, port the C codes to the DSP assembly language. Years ago, this is done manually and thus is a painstaking task. As the modern DSP compilers become more mature, they can do part of the laborious work of instruction selection, parallelizing, pipelining, and register allocation. However, we still often find that the compilers are making mistakes from time to time. In addition, in order to make the final code more compact in size and faster in speed, the C codes have to be tuned to match the DSP architecture. Fig. 4 shows the typical three-step DSP code development flow [41]. With the help of DSP compiler and optimization tools provided by the venders, the programmer can now focus on high level algorithm development first, and then further fine-tune the DSP codes only when necessary.

For porting to DSP, the data type shall be first considered since the definition of data such as integer can be different for different processors. For example, on TI C6000, the long integer means 40 bits. Since the H.264 codec deals with 8-bit pixels, the programmer can use the short data type for fixed-point multiplication, which takes only one cycle. Further optimizations shall make maximal use of all the hardware resources in the critical loops.

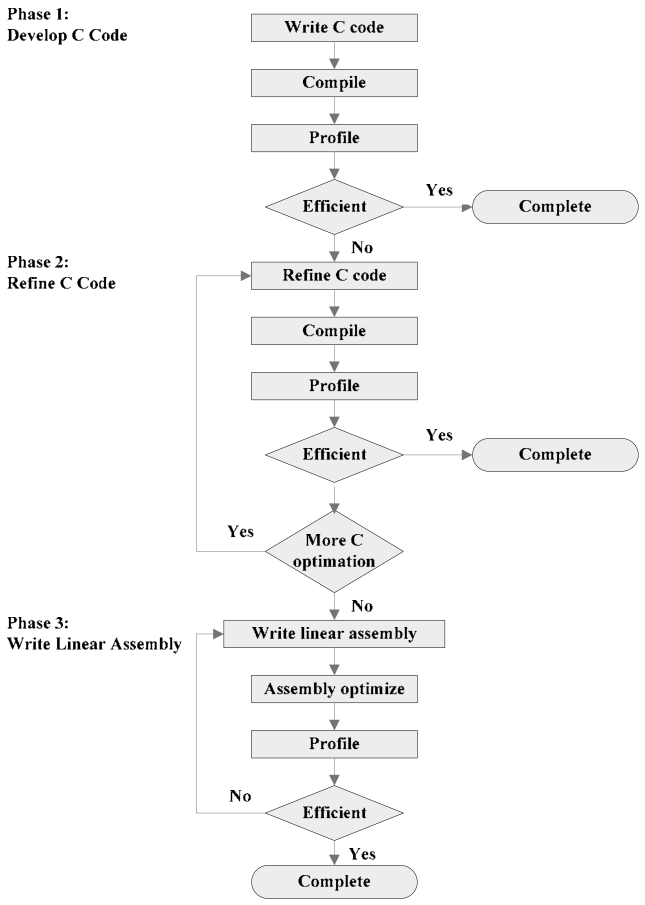

Fig. 4 DSP code development flow

B. *Optimization for DSP architecture*

To maximize C/C++ performance, the following optimization methods can be used [41].

1) *C/C++ language level optimization*

Since the DSP processor has less hardware resources than the PC environment, unnecessary operations has a strong impact on the processing speed. To achieve the best software performance, we use programming tricks to speed up the software at C/C++ language level. For example, we can use look-up table to reduce the arithmetic operations in following decoding steps: inverse transform, de-quantization, and entropy decoding [42].Also, to improve the loop operation, loop unrolling and software pipelining are exploited.

Loop unrolling eliminates or reduces loop management overhead by "unrolling" the loop. Unrolling loops involves replacing iterations of the loop by creating additional copies of the loop itself. Often it leads to faster but larger codes. However, the trade-off between code size and execution performance should be carefully balanced. Unrolling only speeds up code to a certain point, i.e., the law of diminishing returns prevails.

Software pipelining is used to schedule instructions from a loop so that multiple iterations of the loop execute in parallel. In C6000 compiler, the programmer can use "-o2" and "-o3" compiler options. The compiler then attempts to arrange software pipelines for the codes with the information that it gathers from the program.

2) *Intrinsic operator*

The C6000 compiler provides intrinsics, special functions that map certain high-level operations directly to the inline C6x instructions to speed up the C codes. All instructions that are not easily expressed in C codes are supported as intrinsics [41]. For example, we can use the intrinsic operator "_abs" to calculate the saturated absolute value.

3) *Wider memory access for smaller data width*

In order to maximize data throughput, it is often desirable to use a single load or store instruction to access multiple data values consecutively located in the memory. For example, C6x have instructions with associated intrinsics, such as "_add2()", "_mpyhl()", "_mpylh()", etc, that operate on the 16-bit data stored in the high and low parts of a 32-bit register. When operating on a stream of 16-bit data, we can use word accesses to read two 16-bit values at a time, and then use another C6x intrinsic to operate on the data. In the ideal case, we like to get all the units simultaneously operating on all individual instructions. This parallelism is still hard to achieve by the compiler and may still need hand–code in some cases.

4) *Memory management*

To maximize the processing speed, we prefer using internal memory to store instructions and data. However, the internal memory is generally quite small on the DSP processor. For example, on TMS320DM642 processor, there are only 16 Kbytes program memory and 16 Kbytes data memory in L1 cache, and 256 Kbytes unified memory in L2 cache. Therefore, memory management becomes very important. In managing the program memory, we need to delete unused codes and re-write certain functions to decrease the program code size. Next, we can use the compiler options to optimize the execution speed. In managing the data memory, we put all dynamically allocated memory sections into the external SDRAM and put the frequently used data in the internal data memory. This highly efficient memory management can effectively reduce memory stalls.

VIII. Conclusions

H.264/AVC is an efficient video compression scheme but this codec, particularly the encoder, has a very high computational complexity. After a short introduction to the H.264 standard, this paper summarizes a number of existing fast algorithms that can potentially be implemented on a DSP-like processor. These algorithms are mainly aiming at accelerating the computational speed of motion estimation, intra prediction and mode decision modules. In addition, we also describe the typical tricks used to speed up the C codes running on DSP.

Acknowledgements

This work was partially sponsored by ZyXEL Communications Corp., Taiwan, R.O.C., under Grant NCTU-94C031.

References

[1] ITU-T Rec.H.264, ISO/IEC 14496-10 "Advanced video coding", Final Draft International Standard, JVT-G050r1, Geneva, Switzerland, May 2003.

[2] T. Wiegand, G. J. Sullivan, G. Bjontegaad, and A. Luthra, "Overview of the H.264/AVC video coding standard", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, pp. 560-575, July 2003.

[3] C.-L. Yang, L.-M. Po, and W.-H. Lam, "A fast H.264 intra prediction algorithm using macroblock properties," in *Proc. ICIP,* vol. 1, pp. 461 – 464, Oct. 2004

[4] Y.-K. Lin and T.-S. Chang, "Fast block type decision algorithm for intra prediction in H.264 FRext," in *Proc. ICIP*, Oct. 2005

[5] B. Meng, O.C. Au, C.-W. Wong, and H.-K. Lam, "Efficient intra-prediction algorithm in H.264," in *Proc. ICIP*, vol. 3, pp. 837-840, Sept. 2003.

[6] F. Pan, X. Lin, S. Rahardja, K. P. Lim, Z. G. Li, G. N. Feng, D. J. Wu, and S. Wu, "Fast mode decision algorithm for JVT intra prediction," JVT-G013, 7th JVT Meeting, Pattaya, Thailand, March 2003.

[7] Y.-D. Zhang, F. Dai, and S.-X. Lin, "Fast 4x4 intra-prediction mode selection for H.264," in *Proc. ICME*, vol. 2, pp.1151 – 1154, June 2004.

[8] C. C. Chen, T. S. Chang, "Fast three step intra prediction algorithm for 4x4 blocks in H.264," in *Proc. ISCAS*, 2005.

[9] M.-C. Hwang, J.-K. Cho, J.-H. Kim, and S.-J. Ko, "A fast intra prediction mode decision algorithm based on temporal correlation for H.264," in *Proc. of 2005 Int'l Tech. Conf. on Circuits Systems, Computers and Communications,* vol. 4, pp. 1573-1574, Jeiu, July 2005.

[10] J. Jain and A. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, Vol.29, (12), pp. 1799–1808, 1981.

[11] R. Li, B. Zeng, and M.L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, (4), pp. 438–443, 1994

[12] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Trans. Commun.*, 38, (7), pp. 950–953, 1990.

[13] O.T.-C. Chen, "Motion estimation using a one-dimensional gradient descent search," *IEEE Trans. Circuits Syst. Video Technol.*, 10, (4), pp. 608–616, 2000

[14] L.-K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Trans. Circuits Syst. Video Technol.*, 6, (4), pp. 419–422, 1996.

[15] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, 6, (2), pp. 313–317, 1996.

[16] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *IEEE Trans. Image Process.*, 9, (2), pp. 287–290, 2000

[17] C.-H. Cheung and L.-M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, 12, (12), pp. 1168–1177, 2002

[18] C. Zhu, X. Lin, and L.-P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, 12, (5), pp. 349–355, 2002

[19] L. Luo, C. Zou, X. Gao, and Z. He, "A new prediction search algorithm for block motion estimation in video coding," *IEEE Trans. Consumer Electron.*, 43, (1), pp. 56–61, 1997.

[20] Y.-T. Roan and P.-Y. Chen, "A fuzzy search algorithm for the estimation of motion vectors," *IEEE Trans. Broadcast.*, 46, (2), pp. 121–127, 2000

[21] J.H. Lee and J.B. Ra, "Block motion estimation based on selective integral projections," *Int. Conf. on Image Processing*, vol. 1, pp. 689–692, Sept. 2002.

[22] C.-M. Kuo, C-P. Chao, and C-H Hsieh, "A new motion estimation algorithm for video coding using adaptive Kalman filter," *Real-Time Imaging*, 8, pp. 387–398, 2002

[23] J. Chalidabhongse and C.-C.J. Kuo, "Fast motion vector estimation using multiresolution-spatio-temporal correlations," *IEEE Trans. Circuits Syst. Video Technol.*, 7, (3), pp. 477–488, 1997

[24] H.-S. Wang and R.M. Mersereau, "Fast algorithms for the estimation of motion vectors," *IEEE Trans. Image Process.*, 8, (3), pp. 435–438, 1999

[25] K. Lengwehasatit and A. Ortega, "Probabilistic partial-distance fast matching algorithms for motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, 11, (2), pp. 139–152, 2001

[26] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. Image Process.*, 4, (1), pp. 105–107, 1995

[27] S.-M. Jung, S.-C. Shin, H. Baik, and M.-S. Park, "New fast successive elimination algorithm," *Proc. 43rd IEEE Midwest Symp. on Circuits and Systems*, vol. 2, pp. 616–619, Aug. 2000

[28] X.Q. Gao, C.J. Duanmu, and C.R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Trans. Image Process*., 9, (3), pp. 501–504, 2000

[29] S.-M. Jung, S.-C. Shin, H. Baik, and M.-S. Park, "Efficient multilevel successive elimination algorithms for block matching motion estimation," *IEE Proc., Vis., Image Signal Process.*, 149, (2), pp. 73–84, 2002

[30] M. Yang, H. Cui, and K. Tang, "Efficient tree structured motion estimation using successive elimination," *IEE Proc. Vis., Image Signal Process.*, Vol. 151, No. 5, Oct. 2004

[31] H.-M. Wong, O. C Au, and A. Chang, "Fast sub-pixel inter-prediction – based on the texture direction analysis," Proc. *IEEE International Symposium, Circuits and Systems*, Oct. 2005.

[32] C.-C. Cheng, Y.-J. Wang, and T.-S. Chang, "A fast fractional pel motion estimation algorithm for H.264/AVC," in *Proc. VLSI/CAD Conf.*, 2005.

[33] Z. Chen, P. Zhou, and Y. He, "Fast motion estimation for JVT", JVT G-016, 2003

[34] X. Yi, J. Zhang, N. Ling, and W. Shang, "Improved and simplified fast motion estimation for JM," JVT P-021, Oct. 2005

[35] K.-H. Han and Y.-L. Lee, "Fast macroblock mode determination to reduce H.264 complexity," IEICE Trans. Fundamentals, Vol.E88–A, 3, pp.800-804, March 2005

[36] J. Lee and Y. Jean, "Fast mode decision for H.264", LG Electronics Inc, Digital media research laboratory

[37] Z. Zhou and M.-T. Sun, "Fast macroblock inter mode decision and motion estimation for H.264/MPEG-4 AVC," *IEEE International Conference on Image Processing*, Oct. 2004.

[38] P. Yin, A. M. Towropes, and J. Boyce, "Fast mode decision and motion estimation for JVT/H.264," *Pro. ICIP*, pp.853-856, 2003

[39] A. C. Yu and G.R. Martin, "Advanced block size selection algorithm for inter frame coding in H.264/MPEG-4 AVC," *Proc. ICIP*, pp. 95-98, 2004

[40] C. Grecos and M.Y. Yang "Fast inter mode prediction for P Slices in the H264 video coding standard," *IEEE Trans on Broadcasting*, Vol. 51, 2, pp.256-263, June 2005.

[41] Texas Instruments, *TMS320C6000 Programmer Guide*, 2001.

[42] S.-W. Wang, Y.-T. Yang, C.-Y. Li, Y.-S. Tung, and J.-L. Wu, "An optimization of H.264/AVC baseline decoder on low-cost TriMedia DSP processor", *Proc. of 49th SPIE Annual Meeting*, 2004.

.