A Global Approach For MPEG-4 AVC Encoder Optimization

Sajjad Khawar[1], Tran Son-Minh[2], Barthou Denis[1,3], Charles Henri-Pierre[1], Preda Marius[2], and Preteux Fran coise[2]

[1] Université de Versailles, Saint Quentin en Yvelines, France
[2] Institut National de télécommunication, Évry, France
[3] INRIA/SACLAY, France

Abstract. According to projection based on hardware evolution, the video bitrate for full HD video encoded in real time with MPEG-4 AVC is expected to reach the 1 Mbps threshold in 2009.
This paper show that it is possible to outperform this goal using a global approach combining algorithmic and code optimization. The MPEG-4 AVC encoder provides lower video bitrate than previous MPEG-2 and MPEG-4 v2 encoders for the same video quality. This is at the expense of a higher computational complexity (10 times more than previous encoders).
The proposed global approach combines algorithmic improvements and original code optimization techniques. The algorithmic parts consists in a fast mode decision algorithm for intra and inter prediction based on spatial correlation information. Intuitively, the mode of prediction is selected via the exhaustive full search manner. When the modes of the neighbors are already decided, the one of the current block can be inherited from the best matched neighbors. A threshold for matching criteria is defined to ensure the rollback to the exhaustive search in the case of high deviation.
The code optimization part consists in dynamic code generation using SIMD instructions and handling data alignment issues. From a generic description of the essential functions of the encoder, many versions are dynamically generated and automatically optimized according to the alignment of the different blocks of data, outperforming usual implementation resorting to a limited number of hand-coded versions.
With the algorithmic optimization, the experimental results show that it is possible to save the analyzing time and therefore the speed of the encoder up to 16% with a slight increment (less than 1%) in bitrate of the compressed contents. With the code optimization, up to 20% of speed-up is achieved for the encoder, compared to hand-tuned vectorized implementations.

1 Introduction

The new H.264 (or MPEG-4 AVC)[4] video coding standard has gained more and more attention recently, mainly thanks to its high coding efficiency. The

encoder complexity, however, is largely augmented. Among all modules in the encoder; the prediction mode decision (including motion estimation ME in the inter mode) contributes most of the complexity, especially when Rate-Distortion Optimization (RDO)[16] is used. The flexibilities of prediction mode - the main factor that exposes a heavy load on H.264 encoder - can be summarized as followings:

- H.264 encoder deploys a tree-structured hierarchical macroblock (MB) partitions with various sizes as encoded entities. The inter-coded 16x16 pixel MBs can be broken into macroblock partitions of sizes 16x8, 8x16, or 8x8. The latter are also known as sub-macroblocks. Sub-macroblock, in turn, can be further divided into partitions of size 8x4, 4x8 and 4x4. Hereafter, we will refer to these seven block types as 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4 respectively. They can be encoded with either inter or intra (Intra mode can be performed on three block types only: 4x4, 8x8 and 16x16). Therefore, we add a prefix inter or intra respectively to block types to clarify the prediction mode in the future reference.
- If a block is inter-coded, multi reference frame (up to 32) is used to search the most matching block. In addition, quarter pel motion vector precision is used.
- If a block is intra-coded, we can make use of several options: Intra4x4 and intra8x8 support 9 modes of intra prediction while intra16x16 support 4 ones.

Choosing the optimal prediction mode among all of the possibilities above is a complex question. The best result can be obtained by the deployment of the RDO framework. That is, each MB must traverse through all possible inter / intra encoding modes. For each mode, the full encoding process is performed to calculate the resulting bits for encoding that MB. The full decoding process is then involved to reconstruct the MB and to derive the resulting distortion. The prediction mode producing the minimum Global Cost (GC) will be selected as the final mode for the actual encoding process of the MB in question. The mathematic formula of GC is the following:

$$GC_{mode}(s,r,MODE|\lambda) = SSD(s,r,MODE) + \lambda \cdot R(s,r,MODE)$$

In the above equation, SSD denotes the Sum of Square Difference between the original source block s and reconstructed one r. MODE indicates a mode out of a set of potential MB modes: SKIP, Inter16x16, Inter16x8, Inter8x16, Inter8x8, Inter8x4, Inter4x8, Inter4x4, Intra4x4, Intra8x8, Intra16x16 and is the number of bits associated with the chosen MODE, including the bits for the MB header, the motion vectors (in the case of inter mode) and all DCT coefficients (except the SKIP mode, where no DCT coefficients are transmitted). λ is the Lagrange multiplier. Note that in the case of inter mode, the RDO model can be deployed one more time in ME to determine the appropriate reference frame

and motion vector (MV). The cost function in this case can be formulated as following:

$$C_{ME}(\vec{m}, REF | \lambda_{ME}) = SAD(s, c(REF, \vec{m})) + \lambda_{ME} \cdot (R(\vec{m} - \vec{p}) + R(REF))$$

where $\overrightarrow{m} = (m_x, m_y)^T$ is the current MV being considered, REF denotes the reference picture, $\overrightarrow{p} = (p_x, p_y)^T$ is the MV used for the prediction during MV coding, λ_{ME} is the Lagrange multiplier for ME, $R(\overrightarrow{m} - \overrightarrow{p})$ represents the necessary bits used for coding MV and R(REF) is the bits for coding REF. The SAD (Sum of Absolute Differences) is computed as:

$$SAD(s, c(REF, \vec{m})) = \sum_{x=1, y=1}^{B_1, B_2} \left| s[x, y] - c[x - m_x, y - m_y] \right|$$

where s is the original video source and c is the motion compensated one based on REF and $\overrightarrow{m}$; B_1 and B_2 are the vertical and horizontal dimensions of the block size, which can be 16, 8 or 4.

Application of RDO framework (even two times) to the selection of prediction mode means performing the fully encoding-decoding chain for each possible mode, each reference frame and MV, which all have a standardized wide range of values to ensure the existence of the optimal candidate. Therefore RDO framework is highly time-consuming.

The profiling of the X264 application yeilds that the SAD and the SATD function families are the most costly. Both consume an average of 20% each, of the application time for different benchmark videos [1]. These two families of functions have the potentiel of exploiting the powerful SIMD instruction sets available on various modern architectures (SSE on pentium, Altivec on PowerPC). In case, the compiler fails to use these SIMD instructions, due to certain issues (memory alignment, data dependencies, etc), one is obliged to use these instructions through assembly language coding or by the use of compiler intrinsics (same is the case of the previous implementations of these functions). The problem is that with each architectural upgrade (e.g. SSE2 to SSE3), these assembly versions have to be re-written to extract the maximum power out of the architecture. Moreover, for an efficient utilization of the SIMD instructions, the memory alignment of data has to be taken into account [9].

We have overcome these issues by making use of a generic runtime code generation scheme that has the capablity to generate various versions of the code at runtime taking into account the memory alignment of data. The code generators (compilettes), being generic in nature, can be redefined according to the different upgrades of an architecture, with great ease. Being at runtime, these compilettes, can make use of high level language constructs (loops, switches) to generate different variants of code that adapt themselves to the runtime data. This *data-based runtime code adaptation*, helps simplify code thus reducing the code complexity and increasing the performance.

Targeting this problem, we propose two approaches to alleviate the encoding complexity caused by the mode decision mentioned above, while maintaining

the coding efficiency: one is related to software implementation and consists in a specialized dynamic run-time compiler able to handle data alignment problems and the second is algorithmic related and consists in selecting a subset of MB for performing the operations.

In the following sections, we will see the algorithmic approach including the previous work and the algorithmic transformations. Then, we take a look at data-based code adaptation, its implementation with the help of compilettes, their application to SAD and SATD and finally we consider the performed experimentation and the results.

2 Algorithmic approach

Our main key-techniques can be summarized as the followings: 1. The generated bits number for arbitrary block encoded with given intra / inter mode is estimated to avoid complete encoding process in evaluation; 2. We process the exhaustive search throughout all block types and reference frames for only a small number of MBs; for the others, the suboptimal reference frame as well as the appropriate mode selection can be safely derived from those of their neighbor blocks. Experiments show that we can drastically save the time for encoding with a small deviation of bitrates and negligible degradation in quality.

2.1 Review of previous works

In the RDO framework, for each possible combination of prediction mode (block size and reference frame for inter-, block size and prediction type for intra mode), not only the real encoding process of the MB (including transformation, quantization and entropy encoding) but also its decoding counterpart (the inverse of all operators in encoding process) must be carried out to evaluate its GC. Therefore the prediction mode with minimum GC can be considered as an optimal selection. The exhaustive evaluation should be applied to all MBs, thus exposes a computational burden far more demanding than any existing video coding algorithm. To reduce this complexity, a number of efforts have been made to reduce the number of candidates to be evaluated with RDO. In the case of intra coding, edge detection with Sobel operator [20] and DCT transform [20] are involved as a pre-process giving hints for selection. A subset of intra modes to be verified with RDO can be also defined from the intra mode of the correspondent MB in the reference frame [10] as well as the neighbor / parent ones in the current frame[14]. In the case of inter mode, flexible threshold is defined for detecting skipped MB [15]. Wavelet transform[8], Sobel operator / SAD[17] are exploited to skip the RDO verification for the small block sizes. Reference frame of the smaller blocks can be hinted from the larger one[18]. New strategies for MV search on a given reference frames are also discussed in[18]. An integrated scheme intra / inter mode selection is discussed on [18], [19],[17].

The common point in the current work is to limit the range of candidates before verifying them with either RDO or its estimation. Only in a special case:

evaluating the SKIP mode or selecting between 2 main intra / inter modes, the early termination is introduced to skip all other possibilities. We show that the early termination can perform well for every prediction mode, provided that the Measure of Conformance (MC) lies within a safe interval. Therefore we can go further than just eliminating the less probable candidates. In many case, we can even point out directly the suboptimal prediction mode.

2.2 Fast mode decision with low computational cost

It makes sense to say that there is a close correlation between the best candidate verified by RDO (probably optimal) and the one providing the minimum Sum of Absolute Transformed Differences SATD (suboptimal). Recent researches , emphasize that relation via experimental results. Also note that the transformation in SATD is the 4x4 Hadamard transform, which is a very fast projection onto the frequency domain . They are the motivations for us to use a cost function C based on SATD to evaluate the encoding method of a MB instead of the RDO itself. Mathematically, the function C can be represented as followings:

$$C(MODE) = SATD(s,c) + R(\vec{m} - \vec{p}) + 2^{Q/6-2} \cdot \left(R(MODE) + R(REF)\right)$$

where Q is the current quantizer applied to the MB, R(x) presents the number of bits necessary to encode the information x (MV, prediction mode and reference frame). For the intra MB, the $R(\overrightarrow{m} - \overrightarrow{p})$ and $R(REF)$ take zero values, the block c then represents the estimation deduced from the current intra mode. The evaluation of function C does not require the reconstructed picture, which necessitates the real encoding and decoding process. It is the key parameter in our proposed technique. For clarification, we use the term Prediction Mode PM applied to a block to imply all together the block size and the intra prediction mode (intra mode) / the position of reference frame (inter mode), which are used to encode that block. We do not deal with MV, therefore MV is detected separately after knowing the reference frame. Hereafter, we call PM suboptimal, if applying that PM to predict the current block, we obtain the minimum cost C. We first describe our technique in the manner of full search via the two following meta functions:

Analyze Intra()

1. Search for the suboptimal intra mode within 4 possible ones, which are applicable to MB. Call its call Cost_intra16x16.
2. For each partition i of size 8x8 ($1 < i < 4$) in the given MB, search for the suboptimal intra mode out of 9 possible modes of block 8x8. Accumulate this cost to a global cost for mode Intra8x8, called Cost_intra8x8.
3. For each partition j of size 4x4 ($1 < j < 16$) in the given MB, search for the suboptimal intra mode out of 9 possible modes of block 4x4. Accumulate this cost to a global cost for mode Intra4x4, called Cost_intra4x4.

4. The smallest one within Cost_intra16x16, Cost_intra8x8 and Cost_intra4x4 decides the final intra mode for the given MB.

Analyze Inter()

1. If MB can be skipped then stop.
2. With block size 16x16, search for the suboptimal reference frame in all of possible reference ones. Call its cost Cost_inter16x16.
3. Repeat Step 2 with block size of 16x8, 8x16, 8x8. The resulting costs are accumulated in the global cost for that MB, called Cost_inter16x8, Cost_inter8x16 and Cost_inter8x8 respectively. Note that each sub-macroblock can be further divided into 8x4, 4x8 and 4x4. Hence, the accumulative Cost_inter8x8 implies the minimum cost (and thus the prediction mode) of the 4 possible partitions 8x8, 8x4, 4x8 and 4x4 applied to the current sub-macroblock.
4. Call Analyse Intra().
5. The smallest cost in Cost_intra16x16, Cost_intra8x8, Cost_intra4x4, Cost_inter16x16, Cost_inter16x8, Cost_inter8x16 and Cost_inter8x8 decides the encoding mode for the current MB.

The function Analyse Intra() is exclusively applied to MBs of slice I, while Analyze Inter() can process on the MBs of slice P and B. If the output of Analyze Inter() is an inter mode, the type of that MB can be either B or P, depending on the fact that the complete list of reference frames is built up from the passed and the future or only from the passed queue. The outlined two functions are the core implementation of [3], demonstrating one way of avoiding the real RDO framework to select the suboptimal encoding mode for a MB.

Our main goal is to further improve this technique by the introduction of the early terminations (ET) in most of its exhaustive searches. That means ET can happen while examining any PMs in step 1, 2 and 3 of Analyze Intra() as well as in step 2 and 3 of Analyze Inter()). The common point of these steps is to traverse through all possible partition-sizes of a MB; at each possible value of block-size, the minimum PM will be exhaustively detected. Note that at any time, if the current block has top and left neighbors, their suboptimal PM are already decided. We consider the following situation: we are checking the current block with a certain PM, which was used as a suboptimal PM to encode its neighbors. In this case, all such PMs (we may have more than one depending on the number of neighbor blocks) will be treated separately as they can potentially trigger ET. The quantity MC is then calculated for each of them, that is the difference between the costs Cs is derived by applying that PM to both neighbor and current block. Note that the cost C of the neighbor block is already known in the process of selecting its suboptimal prediction mode. In fact, the quantity MC means the similarity in prediction-performance of the given mode for 2 neighbor blocks, which are supposed to have a high spatial correlation (the nature of video). In other words, if we have MC small enough, we can safely confirm the correlation between the two blocks. Then the PM, which was chosen as a suboptimal one for the neighbor block, can be also used

for the current one. We then terminate the searching process and skip the rest of the candidates. In the case of high deviation in MC - the consistence between two blocks is invalid - the searching process for minimum cost C is continued for other candidates in the normal manner. If the ET happens in the Step 1 of Analyze Intra or Step 2 of Analyze Inter (found the PM for full MB), we can stop the function immediately. Otherwise, the ET contributes to a rapid cost calculation of the partial block in MB. We define a threshold T for the goodness of MC, which decides whether the ET is applicable. The threshold T ensures the fallback point to avoid the accumulation of error in our algorithm. In the next section, we will discuss the selection of the threshold as well as the performance of the proposed technique.

2.3 Simulation results

The threshold T plays an important role in our technique. With small T, we have a strict matching condition of MC thus less gain in speed. On the contrary, we avoid more exhaustive loops of searching, but the possibility of fake suboptimal PM is also high. In order to determine the appropriate threshold, we define a gain indicator G for each of the threshold T applied to any test video s_i [**?**] as following:

$$G(s_i,T) = \frac{\underset{T}{Max}(Time(s_i,T)) - Time(s_i,T)}{\underset{T}{Max}(Time(s_i,T)) - \underset{T}{Min}(Time(s_i,T))} + \\ + \frac{\underset{T}{Max}(Size(s_i,T)) - Size(s_i,T)}{\underset{T}{Max}(Size(s_i,T)) - \underset{T}{Min}(Size(s_i,T))}$$

Actually G is the combined performance of T on saving time (the first component) and loss in size / compression efficiency (the second component). Both factors are all normalized over a wide range of T from 0 up to 2000 for every source video si. The resulting gains are then averaged for various video sequences si in . Fig. 1 denotes the evolution of the averaged values for intra and inter threshold T. The trend curves for each of mode represent well the compromise between the gaining in time and the loss in compression efficiency. We can consider the first maximum point of the two averaged curves as the optimal T. Therefore we choose $T_{intra} = 480$ and $Tinter = 352$ for the evaluation of our technique.

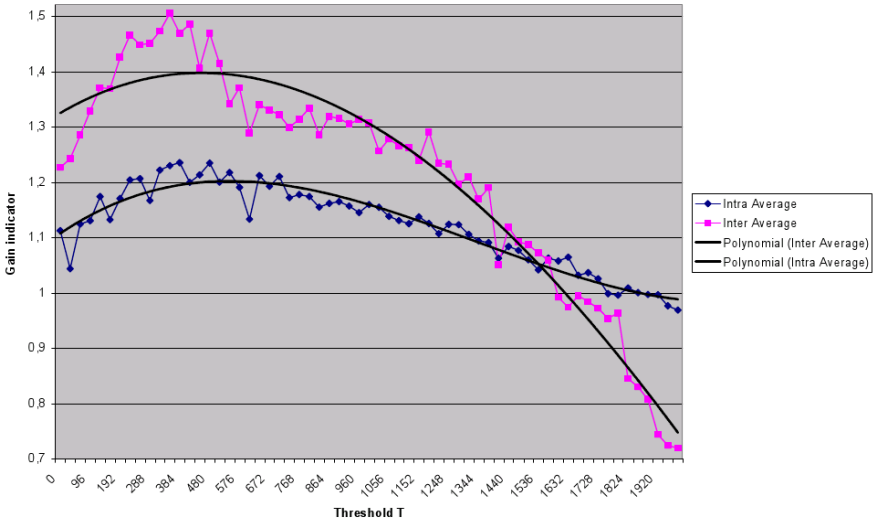

Fig. 1: Evolution of gain indicator in intra and inter mode.

The implementation of our algorithm was integrated into the x264 encoder[**?**], which is considered as one of best H.264 coders concerning its performance on speed and quality[**?**]. Fig. 2 outlines the performance of this coder enhanced with the introduction of our algorithm. In the test process, we deployed the optimal value of T that was determined above.

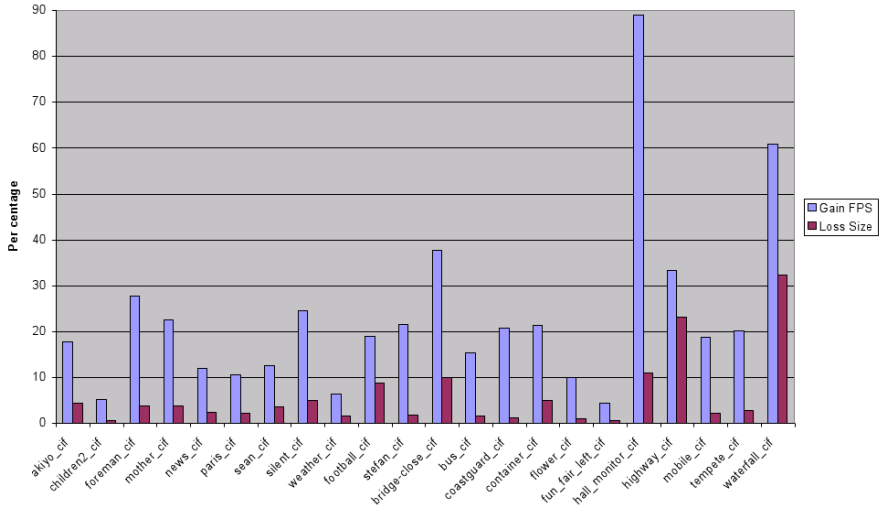

Fig. 2: Overall performance of the proposed technique.

It can be seen that, we can fasten up the encoder on average 23 percent at almost the same quality of video (the lost in PSNR is about 0.08 percent). A loss in compression can be experienced at 5.8 percent on average. In the worst case of the test sequence (highway video), the gain in time is still dominant (over 1.5 times) against the lost in compression.

3 Data-based runtime code adaptation

Program performance is generally effected by its input dataset [5]. Static compilation techniques [6] can address only a limited number of input dataset properties to avoid code explosion. These techniques usually create multiple versions of code for certain dataset properties, on the expense of increased memory usage. They also incur the overhead for the testing of these properties and then for the selection of the correct version at runtime.

Code runtime is the best place for performing a complete analysis of these dataset properties. We can use this information to adapt code according to the dataset. This data-based code adaptation can range from memory locality and memory alignment based adaptations to embedding of dataset values into code as immediate values.

This technique results in a low memory usage and a low version selection overhead, but it usually results in a high code generation overhead. We can reduce significantly this overhead by using compilette-based code generation [section 3.1]. We have used this technique to generate SIMD code for the SAD and the SATD function families for the X264 application.

3.1 Compilette-based generic code generation

Compilettes [7] are generic runtime code generation functions, capable of generating code as simple as one single assembly instruction to a complex sequence of hundreds of instructions.The compilettes can be thought of as a generic layer above the machine code level. They are further divided into two sub-layers. The first represents operations on the algorithmic level, whereas the second represents machine level operations. Once executed, the machine level compilettes generate binary instruction opcodes and store them in a memory buffer. The algorithmic level compilettes mainly control the sequence of execution of the machine level compilettes to generate efficient domain-specific runtime code, at a very small runtime cost. Both these sub-layers can be extended according to the algorithmic domain or the underlying architecture.Moreover, being at runtime, these compilettes can be called from within loops or other control structures. This fact can be used to generate various different versions of the code, based on the runtime values.

Compilettes are supported by a clean and powerful runtime register allocation mechanism. This mechanism helps us organize registers as *register sets*. The register sets can be regarded as compound SIMD variables at the algorithmic level. Whereas at the machine level, one may index into a register set to address a certain specific register within the set. This register representation, being symbolic and compound is more easily manageable as compared to the numeric register representation. The physical register numbers are only assigned at the runtime and are seldom needed at the algorithmic level representation of the program.

Hence, compilette-based code generation can be divided into the two steps: i) expressing the domain specific algorithmic steps in a parametrized way, in

term of machine-level compilettes and ii) using the compilettes thus created, to express the algorithms. One can always experiment very rapidly, with the parameters to extract the best performance out of the underlying architecture.

3.2 Application of data-based code adaptation to X264

As mentioned above [section 1], the SAD and the SATD families of functions are the most time consuming functions in the applications. So, we have focused only on the data adaptation of these functions. Analysis of data shows that these two function families have certain input data characteristics that can be exploited by the compilettes to vectorize the loads and calculations and hence to improve the code ILP. We discuss these characteristics for SAD and SATD separately and also summarize the optimizations that have been performed by the compilettes.

Application to SAD The SAD family of functions computes the *sum of absolute difference* of two blocks of pixels. The code of these functions consists of two regular nested for-loops with known bounds (16, 8 and 4 in this case), and a sum of absolute difference as the only statement in the inner most loop. The functionality can be more precisely expressed as follows.

$$sad = \sum_{i=0}^{N} \sum_{j=0}^{M} abs\left(P_{a_{(i+j*stride_1)}} - P_{b_{(i+j*stride_2)}}\right)$$

The compilers, both GCC and ICC, fail to exploit the powerful SIMD instructions of the IA64 architecture due to the lack of memory alignment information about the two blocks of pixels [P_a, P_b]. They, therefore, resort only to scalar optimizations. As a result the computing power available on this architecture is not well exploited.

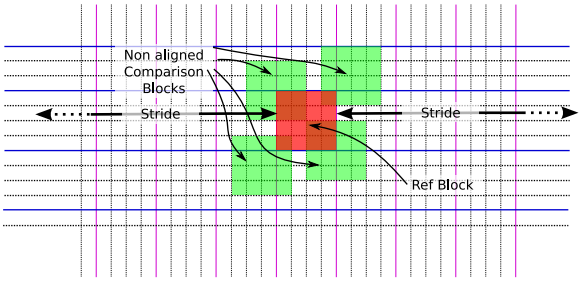

Fig. 1. Memory alignment problem in SAD and SATD

Algorithmic analysis reveals that the first pixel block [P_a] is always aligned at the frontiers of 16, whereas the second block [P_b] has an unknown memory alignment [figure 1]. The compilette-based code generators take into account this

information to vectorize loads. These loads have been complemented with data re-ordering instruction available on the IA64 architecture [12]. The calculations have then been vectorized with the help of the powerful *psad1* instruction. This instruction is capable of calculating the sum of absolute difference of eight pixels from each block, at a time. The knowledge of the loop bounds have been used to completely unroll the two loops. This helps us better exploit ILP on the IA64 architecture.

Application to SATD The SATD family of functions compute the Hadamard transform of two blocks of pixels. The code of this family, is more complex as compared to that of SAD. The functionality of these functions has been given below,

$$satd = \frac{\sum_{i=0}^{N}\sum_{j=0}^{M}\sum_{k=0}^{3}\sum_{l=0}^{3} abs\left(H\left(H\left(P_{sub}^{T}\left(P_{a_{(i*stride_1+4*j)}}, P_{b_{(i*stride_2+4*j)}}\right)\right)\right)^{T}\right)_{kl}}{2}$$

where

$$P_{sub_{ij}}\left(P_a, P_b\right) = P_{a_{(i+j*stride_1)}} - P_{b_{(i+j*stride_2)}}$$

for $i, j = 0, 1, ..., 3$ and

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

The input data $[P_a, P_b]$ has the same properties as that of SAD. Hence, some of the optimizations applied for SAD are similar to those of SATD. So, we do not reconsider what we have already discussed in the above subsection. Here we discuss only the code structure of SATD and the optimizations specific to this family of functions.

Algorithmically, we have four nested loops, with the two inner most loops fixed at 4 iterations each. The expressions also show two transposition operations, that indicate a data-reorganization at the code level. The multiplications with the H matrix, indicate a reduction to add/subtract operations as the matrix contains only 1s and -1s.

In our implementation we have unrolled all the loops in the code. The two matrix transpositions have been optimized by exploiting IA64 special SIMD data-manipulation instructions. The multiplications with the H matrix have been decomposed into SIMD add/subtract operations as mentioned above. Moreover similar operations throughout the code have been merged, where possible, to improve the ILP.

4 Experimentation

We have tested the data-adaptive compilette-based code implementation against all-source-code implementations. Tests have been performed on the kernels and

also on the overall application. We have performed the tests using both GCC and ICC compilers with optimization level -O3. We have obtained upto 7x speedup for the kernels and 20% for the overall application. The details of the experimentation have been presented as follows.

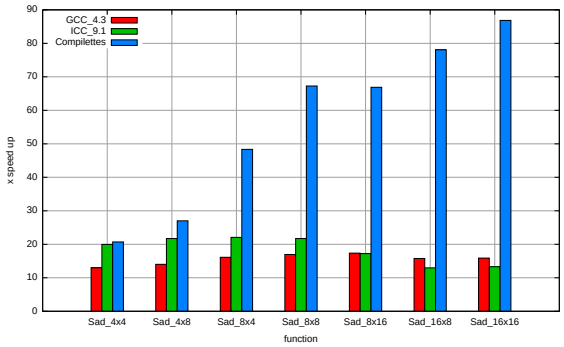

Fig. 2. Performance of different implementations (GCC 4.3 -O3, ICC 9.1 -O3, Compilette) of SAD kernals wrt the C-code version compiled by GCC 4.3 option -O0

4.1 Experimental setup

We have used the X264 application [3] as the test platform. X264 is a free library for encoding H264/AVC [16] video streams. This library is being used by many MPEG-4 applications. Throughout the experimentation we have used either GCC 4.3 or ICC 9.1 to compile the application code. The optimization level has been set to -O3. The performance of the kernels has been shown as the speed up of the respective kernel implementation wrt the all-source-code implementation, compiled by GCC -O0. Whereas, for the overall application performance the application has been compiled once with GCC 4.3 and once with ICC 9.1, with option -O3 and tested against different compilette-powered versions for different benchmark videos. The following hardware experimental setup has been used to perform the experimentation.

Processor: IA64 Itanium2 double core

Frequency: 1.6 GHz

Memory hierarchy as follows:

Memory	Size	Characteristics
L1 (Data)	16 KB	line 64 bytes, load_lat 1 cycle, store_lat 3 cycles
L1 (Instruction)	16 KB	line 64 bytes, load_lat 1 cycle, store_lat 0 cycle
L2	256 KB	line 128 bytes, load_lat 5 cycles, store_lat 7 cycles
L3	3 MB	line 128 bytes, load_lat 14 cycles, store_lat 7 cycles
Main memory	2 GB	

4.2 Comments on experimental results

Before observing the experimental results we must keep in mind that performance of the all-source-code is affected by the compiler used [i.e. GCC or ICC] and on the used optimization level [-O3 in this case]. Whereas the code generated by the compilettes-based code generators does not pass through a compiler and its performance is independent of the compiler. We present our comments on the kernel performances and that of the overall application separately in the following two subsections.

4.3 Kernel Performance

In the performance results, the compilette versions of the kernels outperform the compiler version by far. Secondly the performance of the compilette versions have an increasing trend (wrt the size of the block under treatment) for both SAD and SATD kernels, this is a clear indication of a better ILP exploitation for larger block sizes. This is not the case for the compiler versions of the SAD kernels, where they show no specific trend [figure 2]. This exposes the lack of a good optimization selection strategy on the part of the compilers.

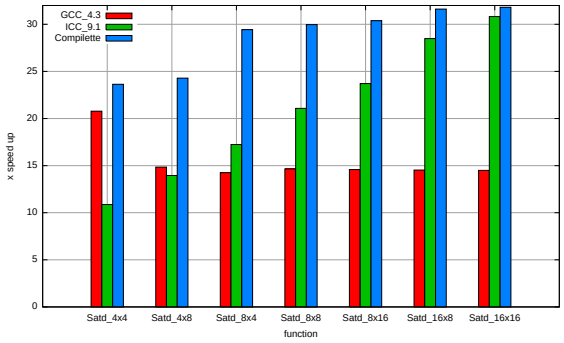

Fig. 3. Performance of different implementations (GCC 4.3 -O3, ICC 9.1 -O3, Compilette) of SATD kernals wrt the C-code version compiled by GCC 4.3 option -O0

On the other hand, the performances for the compiler versions of the SATD kernels show an increasing trend for the ICC compiler and an almost stable trend for the GCC compiler [figure 3]. Moreover the ICC versions achieve a performance very close to that achieved by the compilettes. The ICC compiler implements a single software pipelined version for all the functions. Hence, we observe a good performance when the pipeline is filled. Whereas the GCC compiler optimizes for the smallest size as is evident from the figure 3.It then makes different versions for all other functions using this kernel inside a loop with constant limits. GCC does not make any effort to further unroll this loop or to

optimize separately for each size. As a consequence, the performance is good for the smallest block size but near-constant for the others.

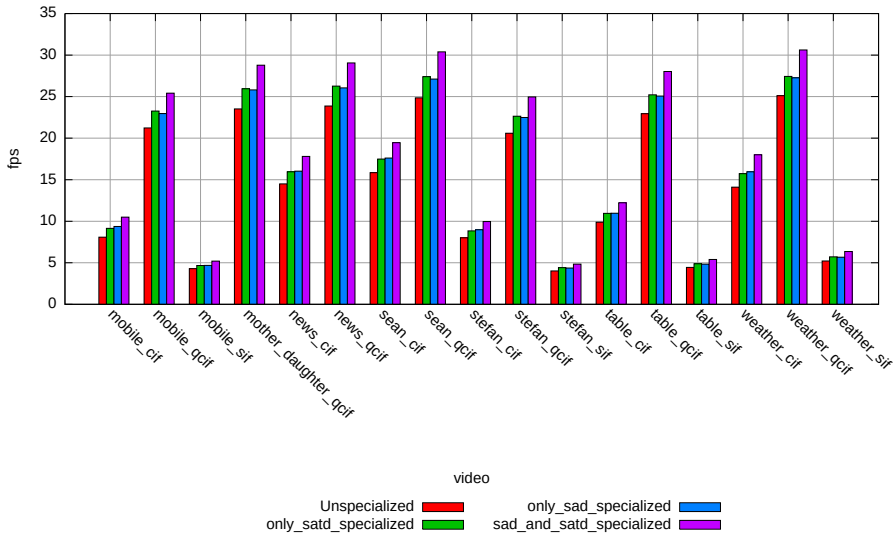

Fig. 4. Overall application performance in frames per second against the C-code version compiled by GCC 4.3 option -O3

4.4 Overall Application Performance

We have integrated the compilette based optimized versions of SAD and SATD in the X264 application. The results have been presented as the frames per second for the different benchmark videos. We have obtained an average speedup of 30% on the overall application wrt to both the GCC [figure 4]and ICC versions [figure 5]. We have also tested the application using one version at a time.

We have observed that the speedup against the GCC versions is due to both the SAD and the SATD kernels. The compilette based versions of both these kernels contribute 50-50 for the overall speedup. Whereas in the case of the ICC compiler, the speedup is mainly due to the contribution of the SAD kernels. This is due to the fact that the ICC version and the compilette version of the SATD kernels resulted in a similar performance for the large block sizes.

5 Conclusions

Targeting the optimization of H.264 encoder, we chose one of the best encoders on the current market as reference system. A cost function is defined to estimate

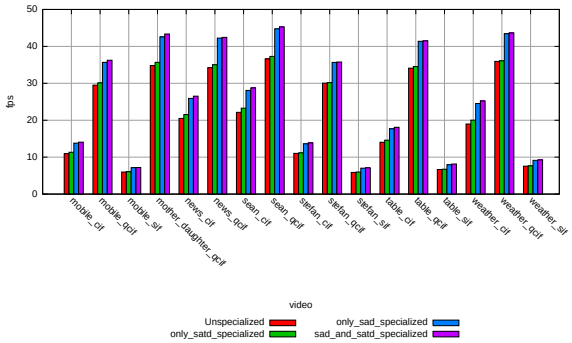

Fig. 5. Overall application performance in frames per second against the C-code version compiled by ICC 9.1 option -O3

as close as possible the RDO framework whenever ME and/or mode decision occurs. To exploit the spatial correlation in the natural video, we introduce a measure of conformance based on the performance of the neighbor prediction mode applied to the currently examined block. If this measurement is less than a safe threshold T, we can reuse the prediction mode for the current block without processing any further checks. We then efficiently avoid a time-consuming exhaustive search. We focus on the determination of the essential threshold T, the key of acceleration as well as the compression efficiency. A common optimal threshold is derived from statistical test on the performance of different T over various types of video.

As the threshold plays a great impact on the performance of our technique, an adaptive and more robust determination of the threshold is among our future researches. The spatial correlation of top left and top right block will be involved in the early termination. We also address the simplification of the CABAC algorithm, the other time consuming factor of the H.264.

We have shown that, unlike hand-written assembly code/compiler intrinsics, the compilettes facilitate the utilization of multimedia/SIMD instructions in a generic way. Being at the runtime they can be utilized in conjunction with high level language constructs (loops/switches) to generate various customized code versions, which is cumbersome when using asm/intrinsics. Moreover, compilettes possess the ability to make use of the real dataset values/properties to further simplify and optimize code. As the compilettes divide the code generation into two logical steps (algorithmic-expression and machine level expression), they can be efficiently and systematically upgraded to future algorithmic/architectural modifications, which is not the case with asm/intrinsics.

In the optimization part we have seen that the data alignment has a major impact on performance and that compiler can not handle it correctly for all cases by static compilation. A dynamic specialized code generator benefits from both run-time knowledge of the data position and the expression power of the run-time computation which allow to easily generate complex code.

In this article we have shown that a two level approach, an algorithmic transformation and a data driven binary code generation, is a realistic approach for the optimization of a real world program (x264). We are able to compress all the MPEG group dataset.

We have shown that both approaches give excellent results. Our future work is to merge both optimizations in the same code source trunc. Our current assumption is that the two optimizations will give an even better speedup, the adaptative algorithmic high level will even reduce the pressure on the low level computing kernels.

Most importantly, this project helps us understand that with small but complex codes (x264 comprise 34K lines of C source code) and treating huge dataset (around 800 Mbytes of video) on a complex architecture like itanium, necessarily needs a global approach to achieve good speedup and good understanding of the performance problems on the high and low level.

References

1. Cif samples video. http://www.tkn.tu-berlin.de/research/evalvid/cif.html.
2. The second annual msu mpeg-4 avc/ h.264 codecs comparison,. http://www.compression.ru/video/codec_comparison/mpeg-4_avc_h264_2005_en.html.
3. X264 open source code project,. http://developers.videolan.org/x264.html.
4. Draft of version 4 of iso/iec 14496-10. Busan Korea, April 2005.
5. N. Drach A. Coveliers, K. Heydemann. Sensibilité aux jeux de données de la compilation itérative. In *In proceedings of SympA Symposium en Architecture de Machine, Perpignan, France*, pages 35–46, 2006.
6. Michel Barreteau, Francois Bodin, Zbigniew Chamski, Henri-Pierre Charles, Christine Eisenbeis, John R. Gurd, Jan Hoogerbrugge, Ping Hu, William Jalby, Toru Kisuki, Peter M. W. Knijnenburg, Paul van der Mark, Andy Nisbet, Michael F. P. O'Boyle, Erven Rohou, Andre Seznec, Elena Stohr, Menno Treffers, and Harry A. G. Wijshoff. OCEANS - optimising compilers for embedded applications. In *European Conference on Parallel Processing*, pages 1171–1175, 1999.
7. Karine Brifault and Henri-Pierre Charles. Efficient data driven run-time code generation. In *In proceedings of Seventh Workshop on Languages , Compilers, and Run-time Support for Scalable Systems, Houston, Texas, USA*, Oct 2004.
8. B-D. Choi, J-H. Nam, M-C. Hwang, , and S-J. Ko. Fast motion estimation and intermode selection for h.264. *EURASIP Journal on Applied Signal Processing*, 2006.
9. Alexandre E. Eichenberger, Peng Wu, and Kevin O'Brien. Vectorization for simd architectures with alignment constraints. *SIGPLAN Not.*, 39(6):82–93, 2004.
10. M-C Hwang, J-K. Cho, J-S. Kim, J-H. Kim, and S-J. Ko. Fast intra prediction mode selection scheme using temporal correlation in h.264. *Proc. of IEEE TENCON 2005, 3D-07.3*, July 2005.
11. C. Kim, H.-H. Shih, and C.-C. J. Kuo. Fast h.264 intra-prediction mode selection using joint spatial and transform domain features. *IEEE International Conference on Image Processing*, Oct 2004.
12. Cameron McNairy and Don Soltis. Itanium 2 processor microarchitecture. *IEEE Micro*, 23(2):44–55, 2003.

13. Project PARA. `http://www.projet-para.uvsq.fr/`.
14. J. S. Park and H. J. Song. Selective intra prediction mode decision for h.264/avc encoders. *Transaction on engineering, computing and Technology,*, May 2006.
15. I. E. G. Ricardson, M. Bystrom, , and Y. Zhao. Fast h.264 skip mode selection using an estimation framework. *Proceedings of PCS 2006*, 2006,.
16. T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuits System and Video Technology*, July 2003.
17. D. Wu, F. Pan, K. P. Lim, S. Wu, Z. G. Li, X. Lin, S. Rahardja, and C. C. Ko. Fast intermode decision in h.264/avc video coding. *IEEE Transaction on Circuit and System for Video Technology,*, July 2005.
18. P. Yin, H. Y. C. Tourapis, A. M. Tourapis, and J Boyce. Fast mode decision and motion estimation for jvt/h264. *IEEE International Conference Image Processing*, Sep 2003.
19. A. C. Yu, G. Martin, , and H. Park. Combined fast intra and inter frame coding for the h.264/avc standard. *31st International Conference on Acoustics, Speech, and Signal Processing*, May 2006.
20. A. C. Yu, G. Martin, and H. Park. A frequency domain approach to intra mode selection in h.264/avc. *Proceedings of 13th European Signal Processing Conference*, Sep 2005.