

# Efficient SIMD-based implementation of adaptive filter

Antti Hallapuro<sup>a</sup>, Dmytro Rusanovskyy<sup>a,b</sup>, Kemal Ugur<sup>a</sup>, Jani Lainema<sup>a</sup>, Moncef Gabbouj<sup>b</sup>

<sup>a</sup>*Nokia Research Center, Tampere, Finland*

<sup>b</sup>*Tampere University of Technology*

**Abstract**—Directional Adaptive Interpolation Filtering (DAIF) is a novel interpolation technique that was proposed recently for hybrid video coding. It was reported, that this technique outperforms the standard H.264/AVC interpolation in terms of coding gain whereas requiring smaller number of arithmetic operations. In this publication we present an optimized implementation of DAIF on a modern computing platform exploiting the Single Instruction Multiple Data (SIMD) parallelism. In addition, we provide a complexity analysis in which the computational complexity is estimated as number of clock cycles per output sample. Proposed SIMD-based implementation of DAIF has lower or comparable interpolation complexity, compared to the highly optimized SIMD-based implementation of the H.264/AVC interpolations. Considering significantly better coding gain provided by DAIF, we believe this approach will play a significant role in future video coding standards.

## I. INTRODUCTION

Most of recent video coding standard, such as H.264/AVC and MPEG-4, are based on hybrid video coding utilizing motion compensation with fractional pixel accuracy. The state-of-the-art video coding standard, H.264/AVC supports use of motion vectors with quarter pixel accuracy for motion compensated prediction. Consider Fig. 1, where integer pixel locations (original image resolution) are noted by upper case letters  $\{A, \dots, F6\}$ , whereas sub-pixel locations are given with lower case letters,  $Sp \in \{a, \dots, o\}$ . In order to estimate and compensate such fractional-pixel displacements, the image signal at the sub-pixel positions has to be estimated by interpolation.

The standard H.264/AVC defines static filters to interpolate values for samples at sub-pixel locations. However, in order to improve coding efficiency of the interpolation, Adaptive Interpolation Filtering (AIF) with Wiener filters has been extensively studied in the recent years [2]-[4]. The AIF schemes proposed in literature differ significantly in terms of filter structure and tap-length. For example, the number of filter coefficients varies from 12 in [4] to 36 in [2] and thus different solutions can have very different complexity. However, all of these schemes provide comparable results on average for a wide set of test video material. It was reported in [2]-[5], that AIF schemes provide improvement of up to 1.1 dB (30% bitrate savings) in coding gain and 17% of bitrate reduction on average for high

resolution video content compared to the standard interpolation filter used in H.264/AVC.

The Directional Adaptive Interpolation Filtering (DAIF) [4] was shown to have the lowest interpolation complexity among the AIF solutions mentioned above, whereas it features a comparable coding gain over the H.264/AVC. The DAIF filter structure allows a platform independent integer 16-bit implementation [5] with complexity comparable to the H.264/AVC interpolation. Such combination of low interpolation complexity and high coding efficiency makes these interpolation algorithms especially attractive for usage in mobile multimedia devices. However, complexity estimates of the DAIF presented in [5] were given as number of arithmetic operations and were not considering any practical implementation or computation architecture.

In this work we port the integer 16-bit implementation of the DAIF onto a selected hardware platform, namely the latest Intel Core 2 processor. This platform provides capabilities for parallel computing at different levels, e.g. using multiple execution cores, multithreading, multiple execution pipelines per core and Single Instruction Multiple Data (SIMD) instructions.

SIMD architecture is nowadays widely used in multimedia applications to accelerate the computations. It uses wide registers and executes several identical operations at the same time within a register. For example, Intel Core 2 SIMD architecture can use its 128-bit registers to hold eight 16-bit words and is able to execute 8 identical operations at the same time with these independent data. Thus, parallelized computations could be implemented 8 times faster.

However, SIMD imposes certain restrictions on parallelized algorithms. For example, result of calculations should not be exceeding dedicated dynamical range, as this would lead to bit-overflow and corruption of data. The platform independent integer 16-bit implementation of DAIF proposed in [5], guarantees that neither final nor intermediate results of calculation exceed the dedicated dynamical range and thus guarantees there is no overflow for the 16-bit size storages. Such property makes this implementation especially well suited for parallel calculations with SIMD architecture.

In this publication we present an optimized 16-bit integer implementation of the DAIF on the latest Intel Core 2 using SIMD instruction set and provide a complexity analysis in

which the computational complexity of the DAIF and H.264/AVC interpolations are estimated as number of clock cycles per output sample. We show that proposed SIMD based implementation of DAIF has comparable complexity to the highly optimized SIMD based implementation of the H.264/AVC interpolation. In the case of 4x4 block based interpolation, SIMD-based DAIF is 14% faster than H.264/AVC interpolation, whereas it requires 16% more clock cycles in the case of 8x8 block interpolation. Considering the significant coding benefits of DAIF, we believe this interpolation scheme has important use cases in mobile multimedia environments where the computational resources are severely constrained.

This paper is organized as follows. Integer 16-bit implementations of H.264/AVC and DAIF interpolations are briefly described in Section 2. SIMD based implementation of DAIF and utilized complexity estimation methodology is presented in Section 3. Section 4 presents complexity estimates for a non-SIMD and SIMD implementations, as well as a complexity comparison between DAIF and H.264/AVC interpolations. Section 5 concludes the paper.

## II. OVERVIEW OF QUARTER-PIXEL INTERPOLATION METHODS

The H.264/AVC defines invariant interpolation filtering to estimate luminance samples at fractional-pixel locations  $Sp \in \{a, \dots, o\}$ , see Fig 1. In order to make this interpolation possible with 16-bit integer arithmetic, the H.264/AVC interpolation is done through a 2-step filtering process. In the first step, samples at all half-pixel positions  $\{aa, bb, b, cc, dd, h, j, ee, ff, gg, hh, hh, ii, jj\}$  of the reference image are estimated by the invariant 6-tap Wiener filter with the following coefficients: [1, 5, 20, 20, 5, 1]/32. In the second interpolation step, samples at quarter-pixel positions are interpolated by a bilinear 2-tap filter.

In contrast, the proposed DAIF [4] interpolates each sub-pixel interpolation independently and utilizes a set of 15 independently computed filters for this. Ten of these filters (for sub-pixels  $Sp = \{a, b, c, d, h, l, e, g, m, o\}$ ) are 1D 6-tap filters and 5 are 12-tap filters ( $Sp = \{f, i, j, k, n\}$ ), see Fig. 1. The direction of each 1D 6-tap filter is determined based on the alignment of the corresponding sub-pixel with the integer pixel samples. More details on utilized filter structure can be found in [4].

### A. Integer 16bits implementations

The following procedure is utilized to interpolate image sample at sub-pixel location  $Y(Sp)$ , (floating point arithmetic is assumed):

$$Y(Sp) = \sum_{i=0}^{N-1} Y_i \cdot h_i(Sp) \quad (1)$$

where  $h_i(Sp)$  is sub-pixel specific filter with floating point representation,  $Y_i$  image samples used for interpolation, and  $N$  is a filter tap-length. Considering an integer representation, filter coefficients  $h$  are represented as  $H_i$ , where  $Q$  determines the accuracy for the coefficients:

$$H_i = \text{sign}(h_i) \cdot \lfloor |h_i| \cdot 2^Q + 0.5 \rfloor, \quad (2)$$

and the filtering process is given as:

$$Y(Sp) = \left( \sum_{i=0}^{N-1} Y_i \cdot H_i + 2^{Q-1} \right) \gg Q \quad (3)$$

In order to implement (3) in 16-bit arithmetic and avoid possible overflows for the cumulative sum  $Y(Sp)$ , filter coefficient  $H_i$ , should be represented with no more than 4 and 5 bit accuracy for  $N=12$  and  $N=6$  respectively. Such low accuracy on the filter coefficient representation suites to the first step of the H.264/AVC interpolation (6-tap filtering), however it would significantly reduce the efficiency of the DAIF interpolation [5].

In order to guarantee 16-bit implementation and preserve high coding gain of DAIF, a novel computation structure was proposed in [5]. The approach is described briefly below. Filtering is performed not according to the equation (3), but through computing of the partial triplet sums, as shown in Fig. 1. Thus, two partial sums are required for 6-tap filtering of  $Y(Sp)$ , for  $Sp \in \{a, b, c, d, h, l, e, g, m, o\}$  and they are given as:

$$\text{res1} = \left( \sum_{i=0}^2 Y_i \cdot h_i^Q(Sp) \right), \quad \text{res2} = \left( \sum_{i=3}^5 Y_i \cdot h_i^Q(Sp) \right). \quad (4)$$

Two additional partial sums are required for 12-tap filtering of  $Y(Sp)$ , with  $Sp \in \{f, i, k, n, j\}$ :

$$\text{res3} = \left( \sum_{i=6}^8 Y_i \cdot h_i^Q(Sp) \right), \quad \text{res4} = \left( \sum_{i=9}^{11} Y_i \cdot h_i^Q(Sp) \right). \quad (5)$$

Following this, negative valued partial sums are clipped toward zero and their unsigned integer representation is utilized. This removes the need to utilize one bit for signaling sign of a value, thus giving us one more bit of precision.

In combination with restrictions imposed on filter coefficient  $h_i$ , see details in [5], this computational technique allows 7-bit accuracy of  $H_i$  representation for adaptive 6-tap filter and 8-bit accuracy for adaptive 12-tap filtering guaranteeing 16-bit implementation. As reported in [5], the resulting platform independent 16-bit implementation of DAIF is as efficient as the floating point DAIF when it comes to coding gain and its interpolation complexity is comparable with that of the H.264/AVC interpolation.

## III. PROPOSED SIMD-BASED IMPLEMENTATION

In this section, SIMD based implementation of the DAIF is described for Intel SSE architecture, although the technique could be utilized for other platforms. SSE architecture contains 128-bit wide registers that can hold either integer or floating point variables. For our implementation we need only 8-bit and 16-bit integer data types.

Implementing algorithms for SIMD architectures can be challenging when the algorithm does more than simple FIR-filtering. This is true for H.264/AVC where minimum interpolated block size is only 4x4 pixels and handling such small blocks is inefficient with 128-bit registers. In this paper, interpolation of 8x8 blocks is described. Interpolation for 4x4

blocks is identical except that only four lines are interpolated and 4 excess columns are discarded.

Interpolation can be divided roughly to 3 stages, data fetching, data rearrangement for MAC operations and actual interpolation. These stages will be described below in more detail.

#### A. Data Fetching and rearrangement to triplet

The amount of pixels that need to be fetched from source image and type of rearrangements depend on the sub-pixel position that is being interpolated. For positions where more than one row is needed, 5 of the rows can be reused when interpolated the next row. The figure also shows the pixels that are used for filtering a single pixel.

##### 1) Horizontal Filtering:

For horizontal interpolation, one row of 16 samples needs to be fetched from memory. This can be done with a single 128-bit wide load operation. Subsequently, samples are arranged into triplets by extracting 6 rows of pixels as shown in Fig 2 (a).

##### 2) Vertical filtering

For vertical filtering, 6 pixel rows with 8 samples in each need to be fetched since vertical 6-tap filter spans 6 samples in vertical direction as shown in Fig 2 (b). Vertical filtering does not require further data arrangements as fetched data is already properly aligned for SIMD operations. Note that 6 rows need to be fetched only when filtering the first row of a block. For the following rows, only one additional row is fetched per interpolated row as 5 previously fetched rows are reused.

##### 3) Diagonal filtering

For diagonal filtering, 6 pixel rows with 16 samples in each are fetched. The relevant pixels in rows are then aligned so that they can be processed efficiently with SIMD operations. Data alignment operation is shown in Fig 2 (c). After performing data aligned, filtering can proceed in identical manner to horizontal and vertical filtering.

##### 4) Diagonal-cross filtering

For diagonal-cross filtering 6 rows with 16 samples in each is fetched. Furthermore, data is to be aligned in a similar way as is done in plain diagonal filtering except that total of 4 triplet are generated instead of 2. This operation is illustrated in Fig 2 (d).

#### B. Triplet Processing

Filtering of triplets is identical for each of the sub-pixel locations. For filtering we utilize MPADDUSBW instruction that performs 16 8-bit multiplications and 8 16-bit accumulate operations in single instruction. After processing two or four triplets as required by the specific sub-pixel location, the triplet results are clipped and summed as specified in section 2. Everything is done with SIMD operations resulting in 8 pixel interpolations performed in parallel.

#### C. Complexity analysis methodology for target platform SIMD

To take into consideration a target platform, we estimate the total number of processor cycles required to execute a

filter on a chosen hardware platform. For the non-SIMD analysis, the number of arithmetic operations to interpolate is used in lieu of cycle estimates. To compute the cycle estimate for the SIMD analysis, we count the operations required to perform the filtering and map the operations onto a selected hardware platform, namely the latest Intel Core 2.

It should be noted that the complexity analysis results serve as a lower bound on computational complexity achievable in a practical implementation. In practice, as it was found in [6], the complexity of well-optimized code to be higher than the estimates produced by this analysis. Some reasons for the difference between the theoretical estimates and measured quantities are listed below.

1. Only operations that belong to the core interpolation kernel are included in the complexity analysis. Operations associated with loop and function call overhead as well as initialization operations are not included. In well organized code they are heavily amortized and therefore represent a small fraction of the overall compute.

2. The analysis accounts for neither data nor instruction cache misses.

3. The analysis assumes that operations are perfectly scheduled to the sub-functions so that maximum efficiency permitted by the instruction mix is maintained at all times.

4. The analysis does not account for platform dependent data alignment details.

5. The analysis does not account for special case code, for example, to handle boundary conditions.

## IV. RESULTS

In this section, we present complexity analysis for 16-bit implementations of DAIF both for SIMD and non-SIMD environments. For the SIMD analysis, we map our implementation onto the Intel Core 2. However, it is a relatively straightforward task to map the implementation onto other hardware platforms that have similar SIMD architectures and thereby extend the results to those platforms. It should be noted that some conclusions might not hold for different architectures but the authors do not in general expect significant discrepancies.

#### A. Non-SIMD Implementations

Using our previous complexity estimate analysis given in [5], which is based on theoretical arithmetic operation counts, we estimated the complexity of DAIF and H.264/AVC interpolations. A block based implementations with block sizes 4x4 and 8x8 were assumed, as such block partitions are defined in the H.264/AVC standard. The complexity numbers are given in the first row of the Table 1.

As one can see, that larger block processing size is favoring for the H.264/AVC interpolation due to re-use of the pre-computed samples during the two-step process. The DAIF complexity advantage over the H.264/AVC interpolation is reduced from 58% in the case of pixel-based interpolation [4] to 21.7% in the case of 4x4 block size implementation and to 13.7% for the 8x8 case.

### B. SIMD-based Implementation

The second complexity analysis covers the SIMD-based implementations of the DAIF and H.264/AVC interpolations ported to the Intel Core 2 platform. The complexities were estimated through the methodology described in Section 3. For this analysis, we count all required processor operations such as data fetching, rearranging in addition to arithmetic operations and computed the minimum required number of processor cycles. A summary of the complexity estimates for two processing block sizes is shown in second row of the Table 1.

Proposed SIMD-based implementation of the DAIF has comparable complexity to the H.264/AVC interpolation. In the case of 4x4 block based interpolation, the DAIF has about 14% lower complexity, in the case of 8x8 block based processing complexity is 16% higher. This follows with the initial conclusion that larger block processing size is favoring for the H.264/AVC interpolation due to re-use of the pre-computed samples during the two-step process.

TABLE I. 1: COMPLEXITY ESTIMATES FOR GIVEN IMPLEMENTATION

Average Estimates	DAIF		H.264		DAIF vs. H.264	
	4x4	8x8	4x4	8x8	4x4	8x8
Number of operations	17.7	17.7	22.6	20.5	-21.7%	-13.7%
Number of cycles	2.05	1.27	2.38	1.10	-14.2%	16.0%

### V. CONCLUSIONS

In this publication we present an optimized 16-bit implementation of the Directional Adaptive Interpolation Filter (DAIF) on the Intel Core 2 which exploits the Single Instruction Multiple Data (SIMD) parallelism. In addition, we provide a complexity analysis in which the computational complexity is estimated as number of clock cycles per output sample. Complexity estimates for analyzed implementations show that proposed SIMD-based implementation of the DAIF is comparable in terms of complexity to the complexity of

highly optimized SIMD-based implementation of the H.264/AVC interpolations. Considering significantly better coding gain of the DAIF, we believe this approach would play a significant role in future video coding standards.

### REFERENCES

- [1] ITU-T Recommendation H.264, Advanced video coding for generic audiovisual services, 7 May 2004.
- [2] Y. Vatis, B. Edler, D. T. Nguyen, J. Ostermann, "Motion and Aliasing-Compensated Prediction Using a Two-dimensional Non-Separable Adaptive Wiener Interpolation Filter," In Proceeding of ICIP'05, 2005.
- [3] S. Wittmann and T. Wedi, "Separable Adaptive Interpolation Filter for Video Coding", In Proc. ICIP'08, San-Diego, USA, Oct. 2008.
- [4] D. Rusanovskyy, K. Ugur, J. Lainema, M. Gabbouj, "Video Coding with Pixel-Aligned Directional Adaptive Interpolation Filters", Proc. of IEEE ISCAS, ISCAS 2008, Seattle, USA, May 2008.
- [5] D. Rusanovskyy, K. Ugur, A. Hallapuro, J. Lainema, M. Gabbouj, "Video Coding with Low Complexity Directional Adaptive Interpolation Filters", IEEE Trans. On Circuits and Systems for Video Technology, Vol. 19, No. 9, August 2009.
- [6] M. Horowitz, A. Joch, F. Kossentini, A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis", IEEE Transaction On Circuits and Systems For Video Technology, Vol. 13, No. 7, July 2003.

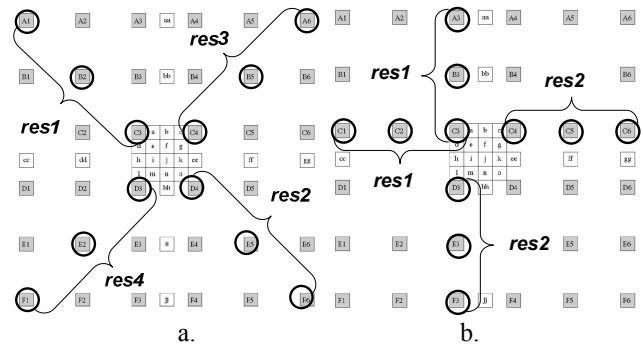


Figure 1. Filter supports of DAIF and computational structure; a) two partial sums (res1 and res2) are used for 6-tap filtering, b) four partial sums used for 12-taps

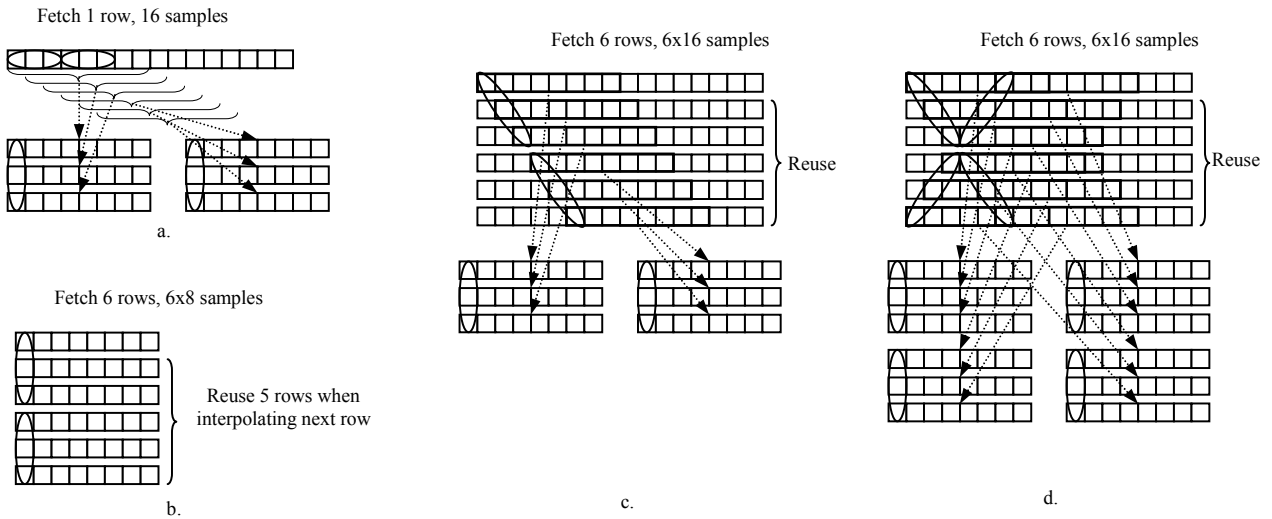


Figure 2. Utilized data fetching and arrangement for SIMD implementation of DAIF