Chi Ching Chi, Mauricio Álvarez-Mesa, Benjamin Bross, Ben Juurlink, Thomas Schierl

SIMD acceleration for HEVC decoding

Suggested Citation
Chi, Chi Ching; Álvarez-Mesa, Mauricio; Bross, Benjamin; Juurlink, Ben; Schierl, Thomas: SIMD acceleration for HEVC decoding. - In: IEEE transactions on circuits and systems for video technology : a publication of the Circuits and Systems Society. - ISSN: 1558-2205 (online). - 25 (2015), 5. - pp. 841-855. - DOI: 10.1109/TCSVT.2014.2364413. (Postprint version is cited, page numbers differ.)

SIMD Acceleration for HEVC Decoding

Chi Ching Chi, Mauricio Alvarez-Mesa, *Member, IEEE,* Benjamin Bross, *Student Member, IEEE,* Ben Juurlink, *Senior Member, IEEE,* and Thomas Schierl, *Member, IEEE,*

***Abstract*—SIMD instructions have been commonly used to accelerate video codecs. The recently introduced HEVC codec like its predecessors is based on the hybrid video codec principle, and, therefore, also well suited to be accelerated with SIMD. In this paper we present the SIMD optimization for the entire HEVC decoder for all major SIMD ISAs. Evaluation has been performed on 14 mobile and PC platforms covering most major architectures released in recent years. With SIMD up to 5× speedup can be achieved over the entire HEVC decoder, resulting in up to 133 fps and 37.8 fps on average on a single core for Main profile 1080p and Main10 profile 2160p sequences, respectively.**

***Index Terms*—H.265, HEVC, SIMD, SSE, AVX, NEON, UHD.**

I. Introduction

Computer architecture and video coding have mutually influenced each other during their technological advancements. The mutual influence is especially strong in the deployment of the single instruction multiple data (SIMD) instructions. SIMD instructions have first been introduced for the purpose of software-only MPEG-1 real-time decoding using general purpose processors [1], [2]. Since then the concept of SIMD has been further exploited by many architectures for many following video coding standards [3]. To allow for more efficient implementation, video coding standards also take the SIMD capabilities of the processors into account during the standardization process. For instance, explicit effort is made to define reasonable intermediate computation precision and eliminate sample dependencies.

The Joint Collaborative Team on Video Coding (JCTVC) has recently released the High Efficiency Video Coding (HEVC) [4] coding standard. HEVC allows for 50% bitrate reduction with the same subjective quality compared to H.264/AVC [5]. Similar to previous standards, significant attention was paid to allow the new standard to be accelerated with SIMD and custom hardware solutions. It is commonly known that hardware-only solutions can potentially provide much higher energy efficiency compared to software solutions using general purpose processors (GPPs). Optimized software solutions, however, are required on platforms where hardware acceleration is not available, have a reduced implementation effort and time-to-market, and avoid hardware overspecialization problems [6].

Many of the improvements to the coding tools responsible for coding efficiency improvements in HEVC over previous standards are also beneficial for accelerating the codec using SIMD. Among others, the larger block sizes, more accurate interpolation filter, and parallel deblocking filter, could make SIMD acceleration even more important than in previous video coding standards.

While HEVC has significant potential for SIMD acceleration, the HEVC standard is also more complex than previous ones. With support for three different coding tree block sizes, more transform sizes, additional loop filter, more intra prediction angles, etc., significantly more effort is required for fully accelerating HEVC using SIMD. This will only become more complex with the addition of future range extensions, which will introduce more chroma formats and higher bit depths. Additionally in recent years much more diversity has been introduced in the instruction set architectures (ISAs) and their micro-architectures. SIMD ISAs have become more complex with multiple instruction set extensions, and even with the same ISA the instructions have different performance characteristics depending on the implementation.

In this paper we investigate the impact SIMD acceleration has on HEVC decoding. For this the entire HEVC decoder has been accelerated, i.e., SIMD has been applied to all suitable kernels and operations. An implementation has been developed for all recent x86 SIMD extensions as well as ARM NEON. The main contributions of this paper are:

- SIMD acceleration is presented for all the data-parallel kernels of the HEVC decoder (main and main10 profiles).
- Implementation and optimization of the HEVC decoder is performed for all relevant SIMD ISAs, including NEON, SSE2, SSSE3, SSE4.1, XOP, and AVX2.
- The effect of an interleaved chroma format on the SIMD implementation is investigated.
- Performance evaluation is performed on 14 platforms providing a large coverage of recent architectures for 1080p (HD) and 2160p (UHD) resolutions.
- With SIMD optimizations the decoder is able to process up to 133 fps for 1080p and 37.8 fps for 2160p on average on recent architectures.

The paper is organized as follows. First, Section II gives an introduction to SIMD ISAs, while Section III presents the related work. Section IV describes the optimized HEVC decoder used as a baseline. Section V presents the SIMD implementation of the suitable HEVC decoding kernels. Section VI details the experimental setup, and in Section VII the performance results are discussed. Finally, in Section VIII, conclusions are drawn.

C. C. Chi, M. Alvarez-Mesa, and B. Juurlink are with the Embedded Systems Architecture group, Technische Universität Berlin. Sekretariat EN 12, Einsteinufer 17, 10587 Berlin, Germany. email: {chi.c.chi,mauricio.alvarezmesa,b.juurlink}@tu-berlin.de

Benjamin Bross and T. Schierl are with the Fraunhofer Heinrich Hertz Institute, Einsteinufer 37, 10587 Berlin, Germany. email: {benjamin.bross,thomas.schierl}@hhi.fraunhofer.de

This work is supported in part by the LPGPU Project (www.lpgpu.org), grant agreement n° 288653.

TABLE I
SIMD EXTENSIONS TO GENERAL PURPOSE PROCESSORS

SIMD ISA	Base ISA	Vendor	Year	SIMD Registers
MAX [8]	PA-RISC	HP	1994	31×32b
VIS	SPARC	Sun	1995	32×64b
MAX-2	PA-RISC	HP	1995	32×64b
MVI	Alpha	DEC	1996	31×64b
MMX [10]	x86	Intel	1996	8×64b
MDMX	MIPS-V	MIPS	1996	32×64b
3DNow!	x86	AMD	1998	8×64b
Altivec [11]	PowerPC	Motorola	1998	32×128b
MIPS-3D	MIPS-64	MIPS	1999	32×64b
SSE [12]	x86/x86-64	Intel	1999	8/16×128b
SSE2 [13]	x86/x86-64	Intel	2000	8/16×128b
SSE3	x86/x86-64	Intel	2004	8/16×128b
NEON	ARMv7	ARM	2005	32×64b — 16×128b
SSSE3	x86/x86-64	Intel	2006	8/16×128b
SSE4	x86/x86-64	Intel	2007	8/16×128b
VSX	Power v2.06	IBM	2010	64×128b
AVX	x86/x86-64	Intel	2011	16×256b
XOP	x86/x86-64	AMD	2011	8/16×128b
AVX2 [14]	x86/x86-64	Intel	2013	16×256b
AVX-512 [15]	x86-64	Intel	2015	32×512b

II. OVERVIEW OF SIMD INSTRUCTIONS

SIMD instructions for GPPs are a variation of the classical SIMD computing paradigm [7]. Their purpose is, as the name implies, to process multiple data elements with the same instruction. In a GPP this is achieved by partitioning each register into subwords and applying the same operation to all of them. With this approach SIMD instructions offer significant performance improvements with relatively little additional hardware. The algorithm, however, must be suited for SIMD acceleration, i.e., it must contain data level parallelism.

SIMD instructions were first introduced for the PA-RISC architecture in 1995 [8] for accelerating MPEG-1 video decoding [9]. After that they have been included in almost all architectures and have been used for accelerating many applications apart from video decoding. One of the main properties of a SIMD ISA is the *SIMD width*, which defines the number of elements that can be processed in parallel within a register. The first SIMD ISAs such as MAX-1 for the PA-RISC, and MMX for x86 used 64-bit registers. The next generation including Altivec for PowerPC, SSE for x86, and NEON for ARM increased the SIMD width to 128-bit. In 2013 Intel introduced a new extension called AVX2 that increased the SIMD width to 256-bit. Recently a new extension specification for the x86 architecture has been released with 512-bit registers called AVX-512. Table I presents an overview of the SIMD extensions released over the years.

Generally a SIMD ISA support arithmetic, logical, load, store, type conversion, and data swizzling instructions. Some SIMD ISAs, however, are more complete than others and support for instance more packed data types or operations. The load and store instructions depend on the SIMD vector width, but some ISAs have stricter rules regarding the data alignment than others. Often there are also differences in the data swizzling instructions that rearrange the data inside a vector. Although SIMD acceleration for most algorithms generally can follow a similar method for different ISAs, implementing an optimal solution for each ISA requires it to be uniquely tuned.

III. RELATED WORK

By using SIMD extensions it was possible in 1995, for the first time with software-only, to decode CIF (352×288) MPEG-1 videos in real-time (25 fps) [9], [1] using a workstation running at 80 MHz. After that, SIMD instructions have been used for accelerating different video codecs such as MPEG-2, MPEG-4 Part 2, and more recently H.264/AVC and HEVC. A summary of works reporting SIMD optimization for codecs before H.264/AVC can be found in [3].

In the case of H.264/AVC SIMD has been used to accelerate luma and chroma interpolation filters, inverse transform and deblocking filter. Using SSE2 complete application speedups ranging from 2.0 to 4.0 have been reported [16], [17]. Real time decoding of 720p content using a Pentium IV processor with SSE3, and a low-power Pentium-M processor with SSE2 have been reported [18].

Some recent works have proposed SIMD acceleration for HEVC decoding. L. Yan et al. [19] have reported a decoder with Intel SSE2 optimization for luma and chroma interpolation filters, adaptive loop filter (not included in the final HEVC standard), deblocking filter and inverse transform. The obtained speedups are 6.08, 2.21, 5.21 and 2.98 for each kernel respectively. The total application speedup is 4.16, taking as baseline the HEVC HM4.0 reference decoder. Using an Intel i5 processor running at 2.4 GHz this system can decode 1080p videos from 27.7 to 44.4 fps depending on the content and bitrate. Bossen et al. [20] have presented an optimized HEVC decoder using SSE4.1 and ARM NEON. On an Intel processor i7 running at turbo frequency of 3.6 GHz the decoder can process more than 60 fps for 1080p video up to 7 Mbps. On a Cortex-A9 processor running at 1.0 GHz it can decode 480p videos at 30 fps and up to 2 Mbps. Bossen has also shown [21] that on an ARMv7 processor running at 1.3 GHz 1080p sequences can be decoded at 30 fps. In this study, however, the experimental setup is not well described direct comparisons cannot be made. Bross et al. [22] have reported an optimized HEVC decoder using SSE4.1 on an Intel i7 processor with an overall speedup of 4.3 and 3.3 for 1080p and 2160p, respectively. When running at 2.5 GHz the system is able to process, in average, 68.3 and 17.2 fps for 1080p and 2160p respectively. Table II presents a summary of the mentioned works reporting SIMD acceleration for video decoders.

In previous works results have been presented for one or two SIMD extensions (such as SSE4.1 and NEON) and one or two processor architectures. Or in some cases only the complete application speedup with SIMD is reported but not the SIMD techniques and the per-stage speedups. Instead in this paper, we present a detailed analysis of the impact of SIMD optimization on the HEVC decoder by comparing implementations for multiple SIMD ISAs. In addition, we quantify the impact of the microarchitecture improvements over several processor generations. Furthermore, we evaluate the chroma interleaved format which benefits SIMD acceleration compared to the traditional planar format. Overall, compared to previous work the performance of the presented decoder is higher when using similar processors and input videos.

TABLE II
VIDEO DECODING WITH SIMD OPTIMIZATIONS

Application	Year	ISA	Processor	Freq. [MHz]	Resol.	fps	S_p
H.264 [16]	2003	SSE2	Pentium-IV	2400	480p	48	3.0
H.264 [18]	2004	SSE3	Pentium-IV	3400	720p	60	n.a.
	2004	SSE2	Pentium-M	1700	720p	30	n.a.
HEVC [19]	2012	SSE2	i7-2400	3400	1080p	28-44	4.2
HEVC [20]	2012	NEON	Cortex A9	1000	480p	30	n.a.
	2012	SSE4.1	i7-3720QM	3600	1080p	60	n.a.
HEVC [21]	2012	NEON	n.a.	1300	1080p	30	n.a.
HEVC [22]	2013	SSE4.1	i7-2920XM	2500	1080p	68.3	4.3
	2013	SSE4.1	i7-2920XM	2500	2160p	17.2	3.3

IV. GENERAL STRUCTURE OF OPTIMIZED HEVC DECODER

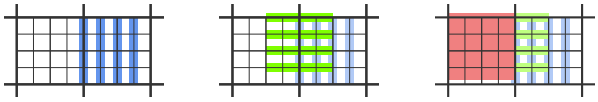
In this section we describe the optimized HEVC decoder used as a baseline for the SIMD optimization. In the description we will focus on the decoding process of a coding tree unit (CTU). We discuss first the steps performed on the CTU level, and then in more detail the parsing and reconstruction which is performed on smaller units such as prediction units (PUs) and transform units (TUs). Afterwards, we present the CTU memory management of the optimized decoder.

A. CTU Decoding

For performance reasons the entire CTU decoding is performed on a small intermediate buffer that has space for the required data window which is slightly more than two CTUs. Part of the kernels can be performed on CTU granularity where others have to be performed on smaller units. Our decoder performs the following high-level steps on a CTU level:

1) *Pre-synchronization*: Before a CTU is decoded a synchronization is performed to check whether the CTU dependencies are resolved. Depending on the parallelization strategy either it is checked if the top-right CTU has been decoded (WPP), or the co-located CTU has been decoded (frame parallel).
2) *Initialization*: In the initialization phase the syntax element data structures are filled with the appropriate neighboring data and initial values. In order to reduce the memory requirements and improve cache performance only the data of current, top and left CTUs is restored.
3) *Parsing and reconstruction*: The CTU syntax parsing, boundary strength calculation, intra prediction, motion compensation, and inverse transform are performed in this step and will be further detailed in Section IV-B.
4) *In-loop filtering*: After the reconstruction, the deblocking and SAO filters are performed. The filters are not fully applied on the current CTU. Due to data dependencies the filters are partially or fully performed on previous decoded CTUs as illustrated in Figure 1 (for luma samples). For deblocking, all vertical edges of the CTU are filtered first (Figure 1a) followed by the horizontal edges (Figure 1b). Due to the deblocking filter specification in HEVC the deblocking of the horizontal edges cannot be fully performed because it requires the deblocked samples from the vertical edges as input. The last 4 sample columns on the right side of the CTU are not available, before deblocking the first vertical edge of the next CTU. The filtering has to be delayed by a minimum of 4 samples to circumvent this issue. In the SIMD accelerated decoder, however, the horizontal edges are delayed half the CTU width (Figure 1b) for better performance. These details will be discussed in Section V-D where the deblocking filter implementation is presented.
   The SAO filter for similar reasons also requires a delay (Figure 1c). Because it requires deblocked samples as input it has to be delayed by 4 samples vertically. In the horizontal direction a full CTU delay is used instead of delaying the minimum of 5 samples to avoid having to cross 4 CTUs with potentially different SAO modes on unaligned data. In the SAO step, the finalized samples are stored to the picture memory.
5) *Post-synchronization*: After each CTU is processed, thread(s) stalled on this CTU are notified. Depending on the parallelization strategy either stalled threads are notified for each CTU (WPP), or the at the end of a CTU line (frame parallel).

Some other management steps such as border exchange, syntax element and sample line buffer management, and picture border extension are performed by the decoder but are left out to simplify the description.



(a) Vertical edges (b) Horizontal edges (c) SAO

Fig. 1. Order and translation of filtering steps to allow CTU based execution. For clarity only the deblocking of edges on a 16×16 grid are illustrated.

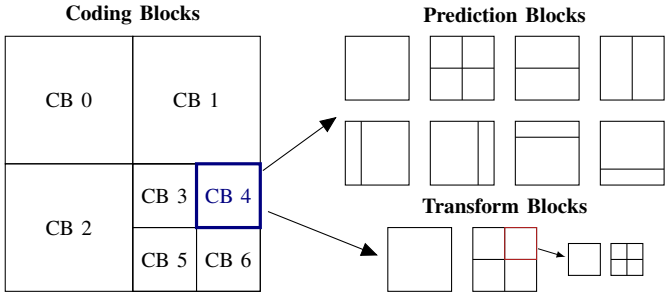


Fig. 2. Subdivision of a CTB in coding blocks (CBs), prediction blocks (PBs), and transform blocks (TBs).

B. CTU Split Process and Leaf Node Processing

In the main parsing and reconstruction step, the CTU is split in CUs using a recursive quad-tree, and the leaf CU nodes are then further split in PUs and TUs. The coding tree block (CTB) subdivision is illustrated in Figure 2. The processing of a final leaf PU is as follows:

1) *Parsing the prediction information*: Depending on the CU type parsing the prediction information involves retrieving the reference indices and motion vectors (inter/skip) or intra luma and chroma modes (intra).

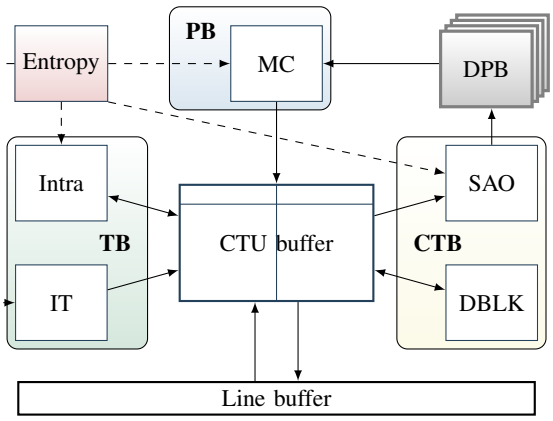


Fig. 3. Overview of the optimized decoder implementation using local buffers. The results of each kernel are stored in the local CTU buffer until they are completely processed.

2) *Motion compensation*: The PB samples for an inter-predicted CB are obtained from an area in a reference picture located by a reference index and motion vector. Interpolation filters (luma/chroma) are used to generate the samples for non-integer positions.

The processing of a final leaf TU is as follows:

1) *Intra prediction*: In HEVC intra prediction is interleaved with the inverse transform on a TU-basis (instead of on a PU basis). Using the mode retrieved for the corresponding PU the intra prediction is invoked.
2) *Parsing coefficients and inverse quantization*: In case coefficients are available for this TU (depends on various conditions), the coefficients are parsed from the bitstream. In our decoder the inverse quantization is performed after parsing the coefficient level and sign. In this way no zero coefficients are needlessly inverse quantized.
3) *Inverse transform*: In case coefficients have been parsed for this TU, the inverse transform is performed to recreate the residual. This step also directly adds the residual to the corresponding prediction (inter or intra) and saturates the result.

C. CTU Memory Management

To reduce the SIMD implementation complexity and improve the cache and memory efficiency the intermediate samples are stored in a local buffer first instead of in the picture memory directly. Figure 3 illustrates the interaction of the different decoding steps and the local buffer. The local CTU buffer can hold the samples for 2 CTUs and their required neighboring samples. The interaction between decoding steps goes via the local buffer. A line buffer is utilized to store and restore intermediate bottom and top neighboring samples, respectively.

This scheme has two main advantages compared to operating on the picture memory directly. First, the SIMD implementation complexity is reduced as only two decoding steps, the motion compensation and SAO filter, are interacting directly with the decoded picture buffer (DPB). Because the decoded samples are either stored in 8-bit or 16-bit containers depending on the bit depth of the sequence, the all decoding functions that interact with the pictures must have two variants.

Second, decoupling the writing to the picture buffer allows for additional memory and cache optimizations. The first writes to the picture buffer introduce significant memory stalls as the output picture memory at this time is not cached. To reduce these stalls CTBs in the picture memory are aligned to cache lines. Modern architectures that support write combining [23] will omit the line-fill read when it detects entire cache lines are written. Aligning this to CTBs ensures that as little cache lines as possible are written to, and consequently are written as fully as possible. Additionally, *non-temporal* store instructions [13] can be used to bypass the cache hierarchy and directly write the lines to memory. This is beneficial because the caches are not polluted with picture buffer data that will not be read until at least the next frame starts decoding. As we will discuss in Section VII-E the usage of non-temporal stores leads to considerable reduction in capacity misses and, consequently, memory transfers.

TABLE III
COEFFICIENTS FOR HALFPEL AND QUARTERPEL POSITIONS FOR THE HEVC INTERPOLATION FILTER.

position	-3	-2	-1	0	1	2	3	4
0.25	-1	4	-10	58	17	-5	1	0
0.5	-1	4	-11	40	40	-11	4	-1
0.75	0	1	-5	17	58	-10	4	-1

V. SIMD OPTIMIZATION

Our HEVC decoder implements SIMD for all the HEVC processing steps except for the bitstream parsing. This includes inter prediction, intra prediction, inverse transform, deblocking filter, SAO filter, and various memory movement operations. We have implemented this for x86, with specialized versions for each of the SIMD extension sets from SSE2 up to AVX2, as well as ARM NEON. For brevity we will not discuss each implementation for every kernel in detail, bu instead we will focus more on the general solutions and challenges involving the SIMD implementations and highlight some distinctions between different instruction sets when required. We will mainly discuss SIMD for the luma component and comment briefly on the chroma implementation.

A. Inter Prediction

During inter prediction, the PB samples must be created from previous pictures indexed by the reference indices. The associated motion vectors specify a translation in these pictures with quarter-sample precision. In case the horizontal or vertical vector component points to a fractional position interpolation is required. HEVC specifies that the interpolation is performed using a 7/8-tap FIR filter of which the coefficients listed in Table III.

Inter prediction is the most time consuming step in HEVC [20]. To derive a horizontally interpolated sample 7 to 8 multiplications and additions must be performed. If also the vertical position is fractional a second filter iteration is applied on the horizontally interpolated samples to derive the final

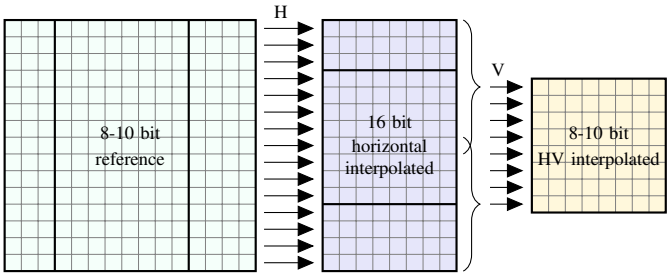
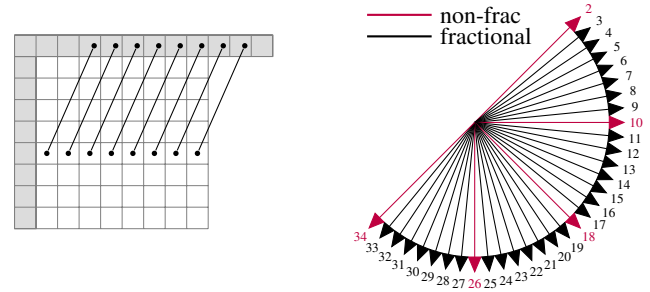


Fig. 4. Horizontal (H) and vertical (V) interpolation of a 8×8 block



(a) Intra prediction (b) Intra angles

Fig. 5. Angular intra prediction in HEVC.

interpolated samples. This process is repeated for the other reference direction in case of bi-prediction and the results are either averaged or weighted to form the final block prediction. The interpolation process is parallel for each sample and is well suited for SIMD acceleration. Figure 4 shows the interpolation process for one direction for a 8×8 block.

While a basic SIMD implementation is straightforward, simply multiplying and adding a vector 8 times either in horizontal or vertical direction, arriving to an optimal solution for each ISA requires more analysis.

- *Input and Intermediate Precision:* Because HEVC supports 8- to 10-bit unsigned input sample value ranges, and up to 16-bit in Range Extension profiles, input samples are either 1 or 2 bytes in size. Furthermore, the interpolation filter gain is between -22 and +88, which adds 8 bits to the intermediate precision. This means that only for 8-bit input precision a 16-bit intermediate precision is sufficient. Because of this implementing a specialized version for 8-bit input bit depth improves performance significantly across all SIMD ISAs.
- *ISA Specifics in Core Computation:* The straightforward implementation of the 7/8-tap filter is to simply perform always 8 multiply-add operations with the different filter coefficients. Closer inspection of the halfpel coefficients show that the filter is symmetric allowing the operations to be factorized, i.e., values corresponding to the same coefficients can be added first before multiplying saving 4 multiplications (often used in H.264/AVC). Also the quarterpel filter requires only 4 multiplications, as the three other multiplications can be implemented with shifts, additions, or subtracts.
  Creating 3 different specialized versions improved the performance for ARM NEON for most of the 8-bit and 16-bit interpolation modes. For the x86 architectures, however, a different implementation is used because there is no regular integer multiply-accumulate instruction. SSE2, however, has the PMADDWD instruction which multiplies the packed 16-bit integers in the source operands and adds the adjacent 32-bit intermediate results. With some rearranging of input values only four PMADDWD and PADDD instructions are required for 4 samples. In SSSE3 also the PMADDUBSW instruction is introduced which is the 8-bit variant of PMADDWD, allowing 8 samples to be computed with only four PMADDUBSW and PADDW instructions. With these instructions no specialization for the halfpel is possible as the factorization would potentially overflow the input vector. We also observed that specialization for the quarterpel filter, using less complex instructions such as shifts and adds, does not improve performance. The PMADDWD and PMADDUBSW are relatively fast instructions and effectively perform more work than a regular SIMD instruction (12/24 ops vs 8/16 ops).
  Additionally specialized versions for different block widths can better utilize the ISAs. In our approach, SIMD versions are created that process 4, 8, or 16 horizontally adjacent samples at a time depending on the block width and available SIMD ISA. For instance, interpolating a 32×32 block can be processed with 8 samples at a time on SSE2, while on AVX2 this can be performed with 16 samples at a time.
- *Multiple Filter Backends:* The inter prediction does not only have different input precisions, but also different output precisions. For instance, HEVC specifies that a horizontal filter that is followed by a vertical filter must not clip the values to the input bit depth, but only shift the values back to fit in a 16-bit range. Furthermore, the different operations must be performed when a single/bi-directional with weighted/averaged prediction is performed. Because this variability exists inside the core loop of the filter, it is not feasible to check this dynamically. Instead an optimized version of the filter is created for each specific case.

B. Intra Prediction

The intra prediction has been refined in HEVC compared to H.264. In H.264 10 distinct modes (dc, planar, 8 angular) of which up to 9 are available depending on the block size. HEVC extends this to 35 modes (dc, planar, 33 angular) which are available to all block sizes (4×4 to 32×32) as shown in Figure 5.

For all modes the derivation of the prediction values is independent, and therefore well suited for SIMD acceleration. For brevity we will focus on the angular modes. Each sample can be derived from extrapolating the position to the boundary samples using the specified angle. If the intersecting position is fractional the prediction value is derived from bilinear filtering two neighboring samples (uses the fractional position as the weight). While each sample can be derived in parallel, several specializations are required for a fast implementation.

- *Fractional and non-fractional angles:* Figure 5b shows that

some angles do not require bilinear filtering. Intra modes 2, 10, 18, 26, and 34 are aligned to 45° and therefore will always hit full boundary positions, and only require a copy of the boundary samples similar to H.264/AVC. For these intra modes a specialized version is created.

- *Vertical angles:* For intra modes 18-34 the samples are extrapolated to the horizontal boundary of the prediction block. This means that all horizontally adjacent prediction samples have the same fractional position, allowing a relatively straightforward SIMD implementation. For the bilinear filtering an intermediate precision of 16-bit can be used for input bit depth up to 10-bit, because the fraction precision is 5 bits and 1 bit is required for the averaging. This means that 8 samples can be predicted in parallel using 128-bit SIMD ISAs.
- *Horizontal angles:* For intra modes 2-17 the samples are extrapolated to the vertical boundary of the prediction block. This means that all vertically adjacent prediction samples have the same fractional position. The prediction can still be performed horizontally, however, by first copying the vertical boundary samples to an array. The prediction samples will then be created in a bottom-to-top, left-to-right order. Storing the produced prediction samples in the right orientation then requires a 90° rotation, which can be implemented using a SIMD transpose and a reverse store of the transposed registers.

While the derivation process of the samples can be accelerated with SIMD, this is not possible for the preparation of the boundary samples. Preparing the boundary samples can be quite complex as samples must be extended for boundaries that are not available for prediction, and afterwards must also filtered. On average, preparing the boundary samples takes about as much time as the prediction

C. Inverse Transform

The inverse transform has traditionally been a well suited kernel for SIMD acceleration. In HEVC this is also true and with block sizes up to 32 × 32 the transform is much more computational complex compared to previous standards [24]. A common implementation of the 2-D inverse transform is to perform a series of two 1-D transforms on the columns and then on the rows of the transform block [25]. The computation of a 1-D column transform consists of a series of matrix-vector multiplication, followed by adding/subtracting the partial results. Figure 6 illustrates the 1-D inverse transform for 32×32 TBs.

The figure shows that for the largest inverse transform the odd positions of the input vector x are multiplied with the 16×16 matrix. Positions 2 to 30 with steps of 4 (4n+2) are multiplied with the 8×8 matrix, and so on. Then the resulting partial solutions are combined with additions and subtractions to the final inverse transformed output vector. In HEVC the smaller 1D-transforms are contained in the larger ones, meaning that the coefficients of the transform matrices are the same for the smaller transforms. The difference when performing a smaller transform is that larger matrix-vector multiplications are omitted and the input position pattern is starting with the odd pattern at one of the smaller matrices.

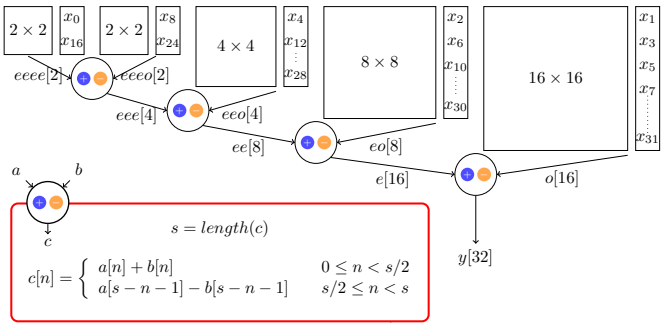


Fig. 6. HEVC 1D inverse transform for 32×32 TBs.

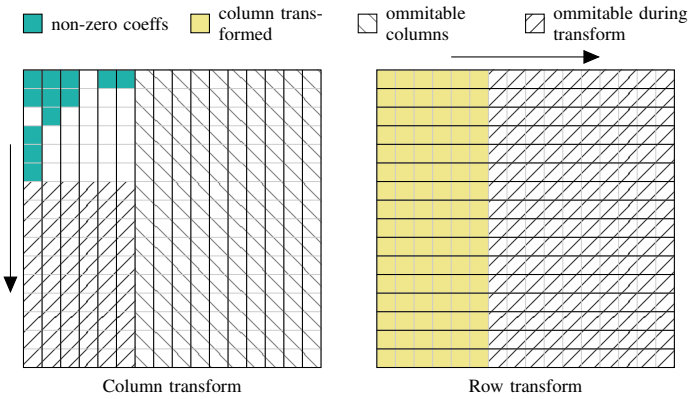


Fig. 7. Omitting unnecessary computation on zero elements in column and subsequent row transform (16 × 16 example).

An optimization that can be performed to the inverse transform in general is omitting calculations depending on the input. It is very uncommon that all positions of the input array are fully populated with non-zero values. Similar to most hybrid video codecs, the coefficient scan pattern of HEVC concentrates the coefficients in the top left corner. Because of the larger transforms in HEVC compared to H.264/AVC more columns and rows contain only zero coefficients for which the computation can be dropped as this would result back in zero. An example of which inputs can be omitted for computation is shown for an 16×16 TB in Figure 7. For the special case that the input coefficients are all zero except for the top left corner position an even more aggressive optimization can be performed, known as the DC transform. In this case the entire inverse transform is omitted as this would result in the same value, which can be computed with a single rounding shift, for all positions in the residual block. These optimizations results in up to 5× speedup for the scalar code, and up to 3.6× and 2.8× for SSE2 and AVX2 respectively for high QP videos.

The SIMD implementation of the inverse transform can be either performed inside one column or row transform or using SIMD on multiple columns/rows, or a combination of the two. We have experimented with these approaches and found that the differences are small. The fastest SSE2 implementation uses SIMD over columns followed by a transpose for both passes of the inverse transform. In this approach not all zero columns can be dropped, because 8 columns are inverse trans-

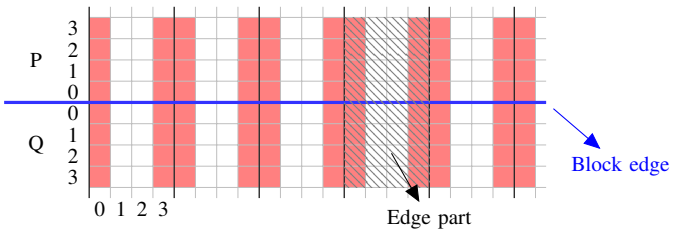


Fig. 8. Samples around a block edge involved in the deblocking filter.

formed at once, and additional transpose overhead is present. This aprroach is still faster, because for the entire transform all the SIMD lanes can be used efficiently which is not the case when applying SIMD to an individual column/row. With NEON the best approach is the same as SSE2 for the smaller transforms (4×4, 8×8). For the larger transforms a combined approach without any transposing overhead proves to be better, in which the first pass used multi-column SIMD and the second pass performs SIMD per row.

D. Deblocking Filter

The deblocking filter can be accelerated using SIMD by considering multiple edge parts simultaneously as illustrated in Figure 8. Because (for <11-bit) the computation precision of the deblocking filter is within 16-bit, 8 computation lanes are available for both SSE2+ and NEON implementations, which can be used to process 4-sample wide edges at a time (for AVX2 16 16-bit lanes are available and can be used to process 8 edge parts simultaneously). A major difference of the deblocking filter compared with other kernels is that a large part of it consists of evaluating conditions [26]. Because multiple edge parts fit inside a SIMD vector the execution of the filter must be made conditional within the vector. The complete SIMD deblocking filter is performed in four phases as described below:

1) *BS check:* In the first phase, if any of 4 consecutive boundary strengths (BS), which are computed during the parsing and reconstruction step, is larger than 0, the samples of 4 edges parts are loaded. For the vertical edge filter these samples are transposed after loading.
2) *Filtering decisions:* In the second phase the first and the last sample column of each of the 4 edge part are swizzled into a 128-bit vector. The following filtering expressions are then evaluated for 4 edge parts at a time.

$$dp0 = |p_{2,0} - 2p_{1,0} + p_{0,0}| \quad dp3 = |p_{2,3} - 2p_{1,3} + p_{0,3}|$$
$$dq0 = |q_{2,0} - 2q_{1,0} + q_{0,0}| \quad dq3 = |q_{2,3} - 2q_{1,3} + q_{0,3}|$$
$$dpq0 = dp0 + dq0 \quad dpq3 = dp3 + dq3$$
$$filter = dpq0 + dpq3 < \beta$$

where $p_{x,y}$ and $q_{x,y}$ are the samples indicated in red in Figure 8, and the value of β depends on the QP of the p and q samples.
3) *Normal/strong filter decision:* If $filter$ evaluates to true for any of the four parts, in the third phase the same swizzled input is used to derive if the filter should be strong or normal. The following expressions evaluate this:

$$strong0 = (|p_{3,0} - p_{0,0}| + |q_{0,0} - q_{3,0}| < \beta/8) \wedge (dpq0 < \beta/8) \wedge (|p_{0,0} - q_{0,0}| < 2.5tc)$$
$$strong3 = (|p_{3,3} - p_{0,3}| + |q_{0,3} - q_{3,3}| < \beta/8) \wedge (dpq3 < \beta/8) \wedge (|p_{0,3} - q_{0,3}| < 2.5tc)$$
$$strong = strong0 \wedge strong3 \quad normal = !strong$$

The results of the first 3 phases are a strong and normal mask, normal second sample mask, and a lossless mask.
4) *Filtering operations:* In the last phase, in which the actual sample filtering takes place, we switch back to the original loaded input and 2 edge parts are filtered at once. If the strong or normal mask is enabled for any of the two edge parts, the computation is performed and the masks are used to select between the filtered samples and the original samples. Finally, samples that are contained in lossless coded CBs should not be filtered independent of previous decisions, and the original samples are selected before storing back in the local CTU buffer.

The SIMD implementation is performing more work compared to the scalar implementation, because multiple edge parts, which might not require the same computation, are processed together. We found that this divergent behavior is stronger if edge parts do not belong to the same block (TB, PB, CB, CTB). The horizontal edge filtering would particularly suffer from this as it must be delayed at the minimum by one edge part in HEVC. Therefore, the horizontal filter is delayed by half the CTB width to reduce the divergent behavior.

E. SAO Filter

The SAO filter has 2 different modes, the edge offset mode and the band offset mode [27]. In both modes the entire CTB is considered and the deblocked samples are used as input. In the edge offset mode 4 sample offsets are transmitted in the bitstream. Each sample $S_{x,y}$ is derived as follows by using the edge offset (EO) type of the CTU, the offsets, and neighboring sample values:

$$S_{x,y} = Clip(P_{x,y} + offset(ind(x,y)))$$

$$offset(m) = \begin{cases} offset_0 & \text{if } m == -2 \\ offset_1 & \text{if } m == -1 \\ 0 & \text{if } m == 0 \\ offset_2 & \text{if } m == 1 \\ offset_3 & \text{if } m == 2 \end{cases}$$

$$ind(x,y) = \begin{cases} s(P_{x,y} - P_{x-1,y}) + s(P_{x,y} - P_{x+1,y}) & \text{if } eo_0 \\ s(P_{x,y} - P_{x,y-1}) + s(P_{x,y} - P_{x,y+1}) & \text{if } eo_1 \\ s(P_{x,y} - P_{x-1,y-1}) + s(P_{x,y} - P_{x+1,y+1}) & \text{if } eo_2 \\ s(P_{x,y} - P_{x+1,y-1}) + s(P_{x,y} - P_{x-1,y+1}) & \text{if } eo_3 \end{cases}$$

$$s(n) = sign(n) = \begin{cases} -1 & \text{if } n < 0 \\ 0 & \text{if } n == 0 \\ 1 & \text{if } n > 0 \end{cases}$$

In the band offset mode also 4 offsets are transmitted in the bitstream. The $index$ derivation is simpler, instead of looking at neighbor samples each sample value is classified in 1 of

32 bands using the 5 MSBs. The 4 transmitted offsets are associated to any 4 consecutive bands, while the other bands will default to an offset of 0.

SIMD can be applied for the entire SAO process by considering multiple samples at the same time, because each sample can be derived in parallel. The $sign$ function can be implemented using clipping to {-1,1} or, when available, with dedicated sign instructions, making the index calculation straightforward. Also the add and clip of the final sample value is possible with all SIMD implementations. The $offset$ can be derived using in register table lookup. For NEON the VTBL instruction can perform 8 lookups at a time, and for SSSE3 and higher 16 lookups can be performed using the PSHUFB instruction. For the x86 processors not supporting SSSE3, however, the lookup has to be performed using regular scalar code.

While the SAO SIMD implementation is straightforward, two considerations must be made. First, for correctness, the samples of lossless coded CBs must not be filtered. Also samples at pictures borders and some slice border must not be filtered. B ecause t hese c ases a re r elatively r are i t i s not desired to add complicated checking inside the inner SIMD loop. We solve this instead by first fi ltering al l th e samples and afterwards put back the original samples where needed. Second, because the SAO is the last step in the decoding process, the output samples are written to the picture buffer and not back to the intermediate buffer. As discussed earlier in Section IV-C, non-temporal stores to the picture buffer are used to reduce cache misses and memory bandwidth requirements.

F. Other Kernels

At various places in the decoder, other than the the highly sequential bitstream parsing, memory operations are required such as filling an array, copying memory, and clearing memory. For example border extensions, setting initial values of syntax elements, and clearing coefficient arrays are also accelerated in our implementation.

G. Chroma Interleaving

Until now we have only considered SIMD acceleration for the luma plane. SIMD acceleration can also be applied to all the kernels for the chroma planes. The chroma planes in some kernels (inverse transform, intra prediction, SAO filter) have (almost) the same derivation process as luma and the code can be shared. In other kernels (deblocking and inter prediction) a different and less complex derivation process is followed, requiring a specialized chroma implementation.

In the HEVC main profile only YUV420 formats are supported. This means that all chroma blocks (TB, PB, CB, CTB) are half the width and height of their luma blocks. Overall this leads to worse SIMD utilization as the smaller blocks are unable to fill the entire SIMD vector.

The SIMD efficiency can be improved by interleaving the two chroma planes horizontally sample-by-sample into one plane with double the width. This semi-planar sample format is referedred to as NV12 for 8-bit and PO10 for 10-bit [28]. In HEVC the SIMD efficiency can be improved using this sample format, because co-located chroma blocks/samples always require the same computation for all the kernels. An exception for this is the inverse transform for which one of the two planes could have no coefficients transmitted. It should be noted that the use of chroma interleaved formats is not restricted to the HEVC codec and 4:2:0 formats but can be applied to any video codec that operates on a planar YUV format.

The usage of chroma interleaved processing does require that the application receiving the output of the decoder must be able to handle the semi-planar color format. The support level for NV12 and P010 is, at the time of writing, not as good as the regular planar formats in many applications. Interleaved chroma formats are currently commonly used in video codec hardware accelerators, but are still rarely supported through the entire displaying software stack.

VI. Experimental Setup

A wide range of platforms has been selected for evaluation, providing a good coverage of the important consumer processor architectures of the last 10 years. The properties of the in total 14 platforms are provided in Table IV , describing the processor and their memory organization . The table groupsthe platforms of the three vendors, Intel, AMD, and ARM. Each platform represents a different core micro-architecture for a generation of processors. Most platforms have multiple cores and in addition most Intel platforms also support simultaneous multithreading (SMT). One AMD platform has also support for hardware multithreading in the form of cluster multithreading (CMT), which promises better performance scaling compared to SMT.

To provide a fair and reproducible evaluation, an as common as possible software stack has been used. All x86 platforms use the Kubuntu 13.04 distribution with Linux kernel 3.8. The ARM platforms run a Linaro distribution that is also derived from the Ubuntu 13.04 packages. For all platforms the GCC 4.8.1 compiler is used with -O3 optimization level. Execution time is measured outside of the program using the `time` command and performance metrics such as instructions, cycles, and frequency are collected with `perf`. For all platforms the dynamic voltage frequency scaling is disabled in all experiments, including the turbo boost and turbo core features of recent processors. The processors are fixed to their nominal frequency listed in Table IV.

Two video testsets have been selected for the experiments. The first include all five 1080p videos from the JCTVC testset [29] and the second one includes five 2160p50 videos from the EBU Ultra-High Definition-1 (UHD-1) testset [30]. All videos were encoded with the HM-10.1 reference encoder using 4 QP points (24, 28, 32, 36). The 1080p sequences are encoded with the random access main (8-bit) configuration, and the 2160p sequences are encoded with the random access main10 (10-bit) configuration. All videos are encoded with wavefront parallel processing (WPP) enabled. Table V shows the resulting bitrates.

TABLE IV
MAIN PARAMETERS OF THE PROCESSORS USED IN THE EVALUATED SYSTEMS.

Architecture	Model	Freq. [GHz]	Cores	MT	L1I/L1D/L2 per core	L2/L3 shared	Process [nm]	Transistors [M]	TDP [W]	Year	ISA
Dothan	LV 758	1.5	1	-	32kB/32kB/-	2MB/-	90	144	7.5	2005	x86
Prescott	Xeon 3.6 GHz	3.6	2x1	2-SMT	12kμops/16kB/-	2MB/-	90	2x169	2x110	2005	x86-64
Atom	D2550	1.86	2	2-SMT	32kB/24kB/-	512kB/-	32		10	2012	x86-64
Conroe	L7500	1.6	2	-	32kB/32kB/-	4MB/-	65	291	17	2006	x86-64
Nehalem	i7-920XM	2	4	2-SMT	32kB/32kB/256KB	-/8MB	45	731	55	2009	x86-64
Sandybridge	i7-2920XM	2.5	4	2-SMT	32kB/32kB/256KB	-/8MB	32	1160	55	2011	x86-64
Haswell	i7-4770S	3.1	4	2-SMT	32kB/32kB/256KB	-/8MB	22	1400	65	2013	x86-64
BayTrail	z3740	1.86[1]	4	-	32kB/32kB/-	2MB/-	22	-	-	2013	x86-64
K8	A64-X2 3800+	2	2	-	64kB/64kB/512kB	-/-	90	150	89	2005	x86-64
Deneb	PII-X4 945	3	4	-	64kB/64kB/512kB	-/6MB	45	758	95	2009	x86-64
Piledriver	FX-8350	4	4	2-CMT	64kB/2x16kB/2MB	-/8MB	32	1200	125	2011	x86-64
Jaguar	A4-5000	1.5	4	-	32kB/32kB/-	2MB/-	28	-	15	2013	x86-64
Cortex A9	Exynos 4412	1.2	4	-	32kB/32kB/-	1MB/-	32	N/A	-	2012	ARMv7
Cortex A15	Exynos 5410	1.6	4	-	32kB/32kB/-	2MB/-	28	N/A	-	2013	ARMv7

[1]Turbo boost frequency used because it could be maintained.

TABLE V
BITRATE (IN MBPS) FOR ALL THE ENCODED VIDEO SEQUENCES.

Video	Hz	Frames	QP24	QP28	QP32	QP36
1080p 8-bit						
BasketballDrive	50	500	10.62	5.16	2.85	1.70
BQTerrace	60	500	19.47	5.58	2.28	1.15
Cactus	50	500	10.55	4.88	2.71	1.60
Kimono	24	241	3.46	1.91	1.08	0.63
ParkScene	24	240	5.43	2.86	1.55	0.85
2160p 10-bit						
FountainLady	50	500	30.82	16.73	8.82	4.78
LuppoConfeti	50	500	24.23	13.79	8.20	5.32
RainFruits	50	500	15.33	8.47	4.97	3.00
StudioDancer	50	500	15.20	8.76	5.13	3.18
WaterfallPan	50	500	38.16	18.55	8.73	4.19

TABLE VI
PERFORMANCE (FPS) 1080P 8-BIT

	QP 24		QP 28		QP 32		QP 36		Gain
	YUV	YC	YUV	YC	YUV	YC	YUV	YC	YC
Dothan	12.4	13.0	16.1	17.0	18.7	19.6	20.6	21.6	5.4%
Prescott	21.0	22.3	29.0	30.9	35.0	37.2	39.7	42.1	6.4%
Atom	11.2	11.8	14.8	15.6	17.4	18.3	19.6	20.5	5.4%
Conroe	22.1	23.2	30.0	31.6	35.9	37.8	40.5	42.6	5.2%
Nehalem	36.2	38.1	51.3	54.2	63.2	66.8	73.0	77.0	5.5%
Sandy Bridge	50.3	53.3	73.1	77.7	92.0	97.8	107.8	114.6	6.1%
Haswell	79.7	86.0	120.4	130.9	155.5	169.8	186.2	203.8	8.6%
Bay Trail	15.8	16.7	21.9	23.1	26.7	28.0	30.5	32.0	5.2%
K8	19.0	20.0	24.5	25.8	28.3	29.7	31.2	32.7	5.1%
Deneb	39.2	41.8	53.9	57.6	64.7	69.2	73.1	78.1	6.8%
Piledriver	57.3	61.7	81.0	87.4	100.0	107.6	115.2	123.7	7.7%
Jaguar	17.1	18.5	23.3	25.3	28.1	30.6	31.8	34.8	8.7%
Cortex-A9	8.1	8.4	10.3	10.5	11.7	11.9	12.9	13.0	2.5%
Cortex-A15	17.5	18.0	22.9	23.2	26.6	26.5	29.5	29.6	1.3%

VII. RESULTS

A. Single-threaded Performance

Tables VI and VII show the performance of a single thread in frames per second for 1080p and 2160p resolutions, respectively. The tables present the performance achieved on each architecture for 4 different QPs. For each QP, both the planar chroma (YUV) and the interleaved chroma (YC) results are shown. The results are averaged over the input videos with the same resolution.

The results show that there is quite a large performance difference for different QPs. Depending on the resolution and architecture, there is a 1.5× up to 2.4× difference between QP 24 and QP 36. Comparing different architectures an even larger performance span can be observed. For instance, there is an up to 15.6× single threaded performance difference between the tested Cortex-A9 and Haswell platform. Most of the tested platforms, however, achieve the common movie frame rate of 24 fps for 1080p. Only the Dothan, Atom, and Cortex-A9 are not able to achieve this on a single core. This is different for 2160p 10-bit as none of the platforms is able to achieve (the expected to be) common frame rate of 50 fps for any QP point, showing the necessity of parallelization.

TABLE VII
PERFORMANCE (FPS) 2160P 10-BIT

	QP 24		QP 28		QP 32		QP 36		Gain
	YUV	YC	YUV	YC	YUV	YC	YUV	YC	YC
Nehalem	11.5	12.0	13.8	14.4	15.8	16.5	17.4	18.2	4.4%
Sandy Bridge	16.1	17.1	19.6	20.8	22.6	24.1	25.3	26.9	6.2%
Haswell	27.1	29.1	33.5	36.3	39.5	43.0	44.4	48.6	8.3%
Bay Trail	4.9	5.1	5.7	6.0	6.5	6.8	7.1	7.5	5.0%
Deneb	11.9	13.6	13.8	16.1	15.4	18.1	16.7	19.8	16.5%
Piledriver	16.8	18.6	19.7	22.2	22.3	25.3	24.4	27.9	12.4%
Jaguar	5.3	5.7	6.2	6.7	6.9	7.6	7.6	8.3	9.3%
Cortex-A9	2.4	2.4	2.7	2.7	3.0	3.0	3.2	3.2	1.2%
Cortex-A15	5.0	5.1	5.7	5.8	6.3	6.4	6.8	6.9	1.5%

Chroma interleaving provides in all cases an improvement. For Intel processors there is a 4.4 to 8.6% improvement and this is stable across resolutions. For AMD at 2160p the improvement is even higher than 16%. On the ARM processors the difference is smaller with up to 2.5% improvement. Chroma interleaving improves the SIMD utilization and memory/cache behavior for the chroma plane(s). For brevity, the results discussed in the next sections use the chroma interleaved configuration.

B. Impact of ISA and Architecture

In the previous section the absolute single-threaded performance was discussed with respect to resolution, QP, and platform. In this section we will refine the platform related results and show the impact of ISA and micro-architectural differences. For all the results, the runtimes of all the videos and QPs are averaged for each resolution.

TABLE VIII
AVERAGE EXECUTED INSTRUCTIONS PER FRAME [MINST/FRAME]

	scalar	neon	sse2	ssse3	sse4.1	avx	xop	avx2
1080p 8-bit								
armv7	319.1	76.5						
x86	385.4		79.1	68.5	67.9	62.7	60.9	49.4
x86-64	344.7		72.0	61.2	60.6	55.9	54.0	42.8
2160p 10-bit								
armv7	1085	299.9						
x86	1250		256.2	243.8	242.3	223.2	210.2	159.7
x86-64	1127		232.9	219.3	218.2	198.9	184.3	138.9

Table VIII shows the average number of instructions executed per frame for the different ISAs and their SIMD extensions. As can be observed, employing SIMD reduces the instruction count dramatically. The instruction count reduction compared to scalar ranges from 4.8× to 8.1× depending on the ISA and SIMD extension. The instruction count reduction is similar for 1080p 8-bit and 2160p 10-bit. For 10-bit typically less scalar instructions are replaced by SIMD instructions, because higher intermediate computation precision is required. This is counterbalanced by the higher resolution, which increases the portion of time spent in kernels that are improved by SIMD.

Comparing the 32-bit and 64-bit x86 architectures shows that 64-bit requires significantly less instructions per frame. This can be mostly accounted to the increased number of architectural registers (8 in x86 and 16 in x86-64), which reduces the generation of so called spilling instructions due to too few available registers. The instruction count for the ARMv7 ISA, which also has 16 architectural general purpose registers, is even lower for scalar execution. NEON, however is less powerful compared to SSE. While both ISAs have 128-bit SIMD instruction, many NEON instructions that perform operations horizontally in the vector, such as table lookup (shuffle), halving adds, and narrowing and widening, are defined only for 64-bit. Also not all SIMD instructions and their options are exposed as intrinsics, putting a higher burden on the compiler for code generation.

In Figure 9 the normalized performance and instructions per cycle (IPC) are shown for 1080p and 2160p, respectively. For the normalized performance, the frequency differences of the platforms are first factored out (by multiplying the runtimes with the frequency), and the results are then normalized to the Haswell scalar results. The plots show how the different architectures would compare to each other when running at the same frequency. It can be seen that SIMD always provides a significant speedup compared to their own scalar baseline. For instance the Atom performs at the same frequency 4.8× slower than Haswell in scalar execution, but is about equal when it uses SIMD. When also using SIMD on Haswell, however, the performance gap returns.

The improvements of the additional SIMD extensions are more incremental, except for avx2 for which the SIMD registers are 256-bit instead of 128-bit. The generation-to-generation architectural improvements have been more significant.

Finally, it can be observed that the IPC when using SIMD instructions are always lower compared to scalar. Because only part of the application is accelerated with SIMD, the IPC of the parts that have more limited acceleration become more dominant. Typically the code parts related to bitstream parsing (CABAC) have low IPC because of frequent branches and data dependencies. Also SIMD code is more optimized and processes data faster leading to relatively more cache misses and control instructions, both typically reducing the IPC.

C. Speedup per Stage

In Table IX the per stage speedup on Haswell is presented for different SIMD extensions averaged over all QPs. For 1080p 8-bit the overall speedup ranges from 3.6× to 4.84× from sse2 to avx2. The stages that improve the most are the inter prediction (10.33×) and the SAO filter (11.18×). The inverse transform (IT) and deblocking filter (DF) benefit less with up to 3.69× and 3.44×, respectively. Although limited, also the PSide, PCoeff, and Other stages are accelerated, mainly from improving filling and clearing memory. (The PSide and PCoeff results are discarded from the table for space reasons.)

Figure 10 shows the execution profile of the decoder for scalar and SIMD for the Haswell platform. It can be observed that portion of time spent on parsing the side information and coefficients (PSide, PCoeff), intra prediction (Intra), and Other increases when using SIMD. The contribution of these stages is relatively higher because the SIMD acceleration has limited effect in these stages.

TABLE IX
SPEEDUP PER STAGE HASWELL FOR DIFFERENT SIMD LEVELS.

	Intra	Inter	IT	DF	SAO	Other	Overall
1080p 8-bit							
sse2	1.23	6.03	2.68	2.80	4.03	1.24	3.67
ssse3	1.30	7.16	2.87	3.00	7.82	1.24	4.14
sse4.1	1.31	7.12	2.98	3.00	8.09	1.24	4.15
avx	1.33	7.58	3.23	3.04	8.26	1.24	4.29
avx2	1.33	10.33	3.69	3.44	11.18	1.18	4.95
2160p 10-bit							
sse2	1.53	5.28	3.34	3.25	3.61	1.16	3.59
ssse3	1.63	5.53	3.34	3.41	4.22	1.16	3.74
sse4.1	1.64	5.51	3.42	3.45	4.27	1.16	3.75
avx	1.65	5.74	3.74	3.49	4.26	1.16	3.85
avx2	1.67	8.03	4.68	4.05	4.75	1.09	4.60

The SAO filter shows the widest speedup span across SIMD extensions. This is especially caused by sse2 which cannot accelerate table lookups. This requires the PSHUFB instruction introduced in ssse3. Apart from avx2 which shows the

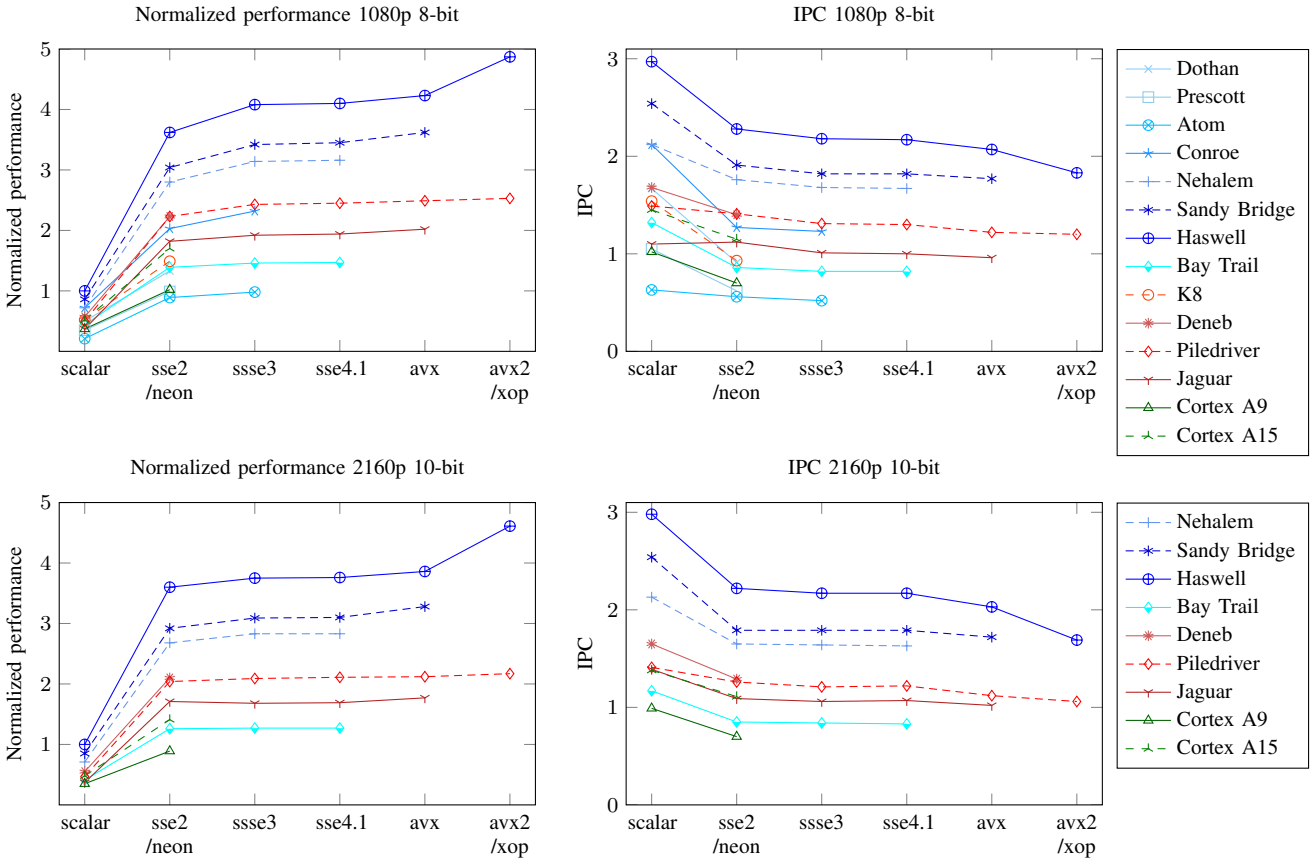


Fig. 9. Single threaded performance at normalized frequency and IPC results.

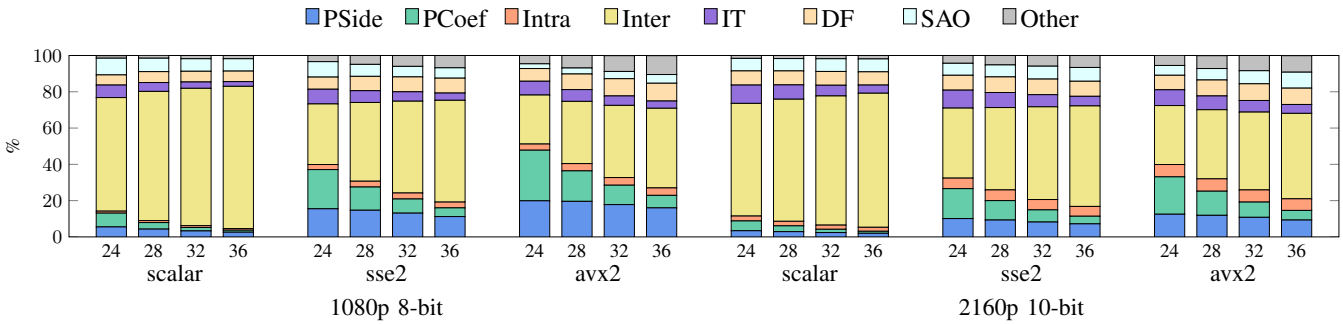


Fig. 10. Decoder execution time breakdown on Haswell for scalar, sse2, and avx2. The PSIDE and PCOEF stages represent the side information and coefficient parsing (CABAC entropy decoding). The PSIDE includes also the interpretation of the syntax elements and coding tree unit traversal.

biggest improvement due to the wider vectors, ssse3 improves the performance the most, mainly due to the introduction of the PSHUFB instruction.

The 2160p (10-bit) results show in general the same trend as the 1080p (8-bit) results. The main differences are that the inter prediction and SAO filter achieve lower speedups, while the intra prediction, inverse transform, and deblocking filter show higher speedup. The inter prediction and SAO filter are the two stages where data is read from and written to picture memory. With 10-bit samples the picture memory and bandwidth per sample is doubled putting more pressure on the memory system. The SAO filter achieves less than half the speedup for 2160p 10-bit compared to 1080p 8-bit and is clearly bottlenecked by the memory system. The impact on inter prediction is less, because the memory accesses are spread over a larger portion of time and can be more effectively overlapped with computation. Also the computation when using 10-bit pictures is more costly, counterbalancing the increased load operations.

The intra prediction, inverse transform, and deblocking filter do not operate on the picture memory directly, but on an intermediate buffer and the increased bit depth has limited consequences in these stages. The speedup for 2160p sequences is slightly higher because of the more common use of larger prediction and transform blocks, which make more effective use of the SIMD vector width. For the deblocking filter larger blocks result in less divergence in the filter evaluation, reducing the amount of redundant work.

TABLE X
BEST PER STAGE SIMD SPEEDUP OVER SCALAR FOR 1080P 8-BIT.

	Intra	Inter	IT	DF	SAO	Other	Total
Prescott	1.08	4.27	2.50	2.08	2.38	1.07	2.91
Atom	1.45	7.21	2.79	2.34	5.36	1.17	4.68
Conroe	1.26	4.76	2.39	2.42	3.95	1.12	3.27
Nehalem	1.30	7.83	2.96	2.96	8.82	1.16	4.50
Sandy Bridge	1.31	7.48	3.10	2.95	8.85	1.20	4.24
Haswell	1.33	10.33	3.69	3.44	11.18	1.18	4.95
Bay Trail	1.34	5.09	2.45	2.42	3.65	1.11	3.30
K8	1.27	3.69	1.84	2.30	3.54	1.11	2.88
Deneb	1.35	6.24	2.50	2.69	3.68	1.22	3.97
Piledriver	1.38	9.48	3.50	2.92	8.77	1.21	5.15
Jaguar	1.40	9.48	2.83	2.65	6.88	1.36	5.44
Cortex-A9	1.45	4.15	1.89	1.78	2.02	1.49	2.82
Cortex-A15	1.42	5.91	2.14	2.21	2.30	1.07	3.27

TABLE XI
BEST PER STAGE SIMD SPEEDUP OVER SCALAR FOR 2160P 10-BIT.

	Intra	Inter	IT	DF	SAO	Other	Total
Nehalem	1.65	5.92	3.42	3.35	4.75	1.11	3.96
Sandy Bridge	1.67	5.78	3.69	3.44	4.27	1.18	3.84
Haswell	1.67	8.03	4.68	4.05	4.75	1.09	4.60
Bay Trail	1.74	4.15	2.82	2.67	3.12	1.04	3.09
Deneb	1.78	5.45	2.97	2.97	3.35	1.20	3.76
Piledriver	1.78	7.14	3.99	3.33	5.34	1.20	4.58
Jaguar	1.94	8.36	3.31	2.95	5.67	1.34	5.17
Cortex-A9	1.74	3.62	2.00	2.01	1.23	1.50	2.55
Cortex-A15	1.89	4.83	2.22	2.50	1.27	1.01	2.91

Finally, Tables X and XI show the speedup per stage for each architecture comparing their best performing SIMD extension to their own scalar baseline. Similar observations as for Haswell can be made for the other architectures. Mostly inter prediction and the SAO filter have the highest speedup, while the improvements for the inverse transform and deblocking filter are more moderate. More memory limited architectures and architectures without a SIMD table lookup instruction, suffer from lower speedups for the SAO filter. Prescott and K8, only support SSE2, while Conroe and the ARM processors are known to be more memory limited. It can also be observed that the overall speedup on older architectures such as Prescott, Conroe, K8, and the ARM processors is lower than on the latest architectures of Intel and AMD. Over the years more emphasis has been put on SIMD performance, because more relevant applications have been optimized for SIMD. An exception is the Bay Trail platform, which improves performance compared to Atom by introducing out-of-order execution. The SIMD execution remained in-order leading to a relatively lower speedup.

D. Multithreaded Performance

Orthogonal to SIMD acceleration, additional performance can be gained by using multithreading. For parallelization we used an approach based on [31] which combines WPP and frame-level parallelism. In this strategy threads decode the rows of a frame in wavefront order using WPP substreams, and additionally, rows of the next frame are already started before fully completing the current frame. To make this approach fully standard compliant motion vector dependencies are tracked dynamically.

Tables XII and XIII show the multithreaded performance for 1080p 8-bit and 2160p 10-bit respectively. The results are averaged over the four QPs. For reproducibility in the experiments the threads are pinned to individual physical cores first, and secondary to the hardware threads exposed by hardware multithreading.

TABLE XII
1080P 8-BIT MULTITHREADED PERFORMANCE

	1	2		4		8	
	fps	fps	S_p	fps	S_p	fps	S_p
Dothan	17.2						
Prescott	31.3	59.7	1.91	70.4	2.25		
Atom	15.9	31.1	1.96	42.7	2.69		
Conroe	32.1	62.1	1.93				
Nehalem	55.0	109.2	1.98	208.0	3.78	223.0	4.05
Sandy Bridge	79.1	155.5	1.97	293.5	3.71	336.6	4.26
Haswell	133.0	259.7	1.95	482.6	3.63	543.0	4.08
Bay Trail	23.5	45.7	1.95	86.8	3.69		
K8	26.1	51.3	1.96				
Deneb	58.4	113.1	1.94	215.2	3.69		
Piledriver	88.9	171.4	1.93	328.7	3.70	483.5	5.44
Jaguar	25.8	50.2	1.94	91.4	3.54		
Cortex-A9	10.7	20.8	1.95	35.5	3.32		
Cortex-A15	23.5	44.6	1.90	77.8	3.31		

TABLE XIII
2160P 10-BIT MULTITHREADED PERFORMANCE

	1	2		4		8	
	fps	fps	S_p	fps	S_p	fps	S_p
Nehalem	14.9	29.5	1.98	57.4	3.86	67.7	4.54
Sandy Bridge	21.6	42.6	1.97	82.7	3.83	98.0	4.54
Haswell	37.8	73.4	1.94	134.9	3.57	155.3	4.11
Bay Trail	6.2	12.2	1.95	23.6	3.79		
Deneb	16.6	31.8	1.92	59.9	3.61		
Piledriver	22.9	43.9	1.91	82.7	3.60	120.5	5.25
Jaguar	7.0	13.2	1.90	23.3	3.35		
Cortex-A9	2.8	5.3	1.90	8.6	3.05		
Cortex-A15	6.0	11.4	1.91	19.9	3.33		

Overall the decoder scales well with multiple physical cores for both 1080p and 2160p. With 2 cores the scaling is mostly close to 2×, while with four cores the speedup is around 3.8×. The ARM cores scale noticeably less, due to a relatively weaker memory system. It can also be observed that Intel's SMT and AMD's CMT do not provide the same improvement as when the threads are executed on individual cores. SMT provides a performance improvement between 7.2% and 37.2% with a typical improvement of ∼15%. CMT fares better with an improvement of 45% to 47.1%, because more resources in the core are duplicated.

E. Memory Footprint and Bandwidth

The maximum resident set size when decoding the random access encoded sequences of the optimized decoder for x86-64 is 29.9 MB and 191.6 MB for 1080p 8-bit and 2160p 10-bit, respectively. When using wavefront and frame-level

parallelism an extra picture buffer is used and the memory footprint rises to 34.1 MB and 222.6 MB, respectively. For ARMv7 and x86 32-bit architectures the memory footprint is slightly lower (1-2%) due to the smaller pointers.

A larger difference between ARMv7 and x86/x86-64 is present in the binary size. For ARMv7 this is 756 KB, while for x86-64 1.92 MB is required. This is mostly caused by the many SIMD extensions of x86 which all are contained in the same binary. During execution only one part of this binary is actually residing in the caches, though, as only one of the SIMD variants is used per sequence.

TABLE XIV
AVERAGE MEMORY TRAFFIC PER FRAME AND BANDWIDTH ON HASWELL.

	MB/frame						MB/s	
	scalar		avx2		4T + avx2		4T + avx2	
1080p 8-bit	read	write	read	write	read	write	read	write
YUV	10.1	3.7	9.6	3.7	7.9	3.7	3978	1903
YC	8.8	3.6	7.5	3.7	6.2	3.7	3321	2045
2160p 10-bit	read	write	read	write	read	write	read	write
YUV	68.2	26.0	48.1	26.4	48.2	26.4	6367	3509
YC	67.0	25.9	41.6	26.5	41.5	26.3	5994	3847

Table XIV shows the average bytes transferred per frame and the memory bandwidth when using scalar, avx2, and multithreading. The bytes written per frame is very similar for all configurations and corresponds closely to the actual data size of the frame with extended borders. The number of bytes read per frame are in all cases higher than the bytes written per frame, which is expected because in the random access configuration the blocks use mainly bi-directional prediction, which reads more than two times the samples it writes. The benefit of using non-temporal stores is clearly visible when comparing scalar to avx2. Especially for 2160p 10-bit the memory transfers savings are significant. The 8 MB large L3 cache can only fit part of the 27 MB picture buffers, and many additional capacity misses are avoided by not write allocating cache lines for the produced picture buffer data. Chroma interleaving also clearly reduces the memory requirements as wider blocks improve the utilization of the data fetched in a cache line.

The average memory bandwidth is significantly higher for 10-bit video because of the 16-bit storage type compared to 8-bit for 8-bit videos. The total memory bandwidth requirements for high framerate (120fps+) 2160p video is high at around 10 GB/s, but is feasible for current mainstream systems which have a practical limit of around 20-25 GB/s.

VIII. CONCLUSIONS

As for previous video coding standards, HEVC is also well suited for acceleration with SIMD instructions. Compared to a well optimized HEVC decoder an additional speedup of 2.8 to 5× can be obtained over the complete decoding process by using SIMD. The acceleration factor provided by SIMD and multithreading allows real-time HEVC decoding to be easily performed on current hardware platforms, even for Ultra High Definition (UHD) applications.

The large speedup, however, could only be achieved with high programming complexity and effort. The complexity of the HEVC standard and the diversity of current computer architectures required many specializations to achieve the optimal performance. Even when introducing a conceptually simple change, such as an interleaved chroma format, a specialized SIMD version for almost every chroma function is required. When video standards and applications continue to increase in complexity, the programmability of SIMD could become the main bottleneck for achieving the highest performance.

To alleviate the programmability issues of SIMD the synergy between the SIMD ISA, the video codecs, and foremost the programming model has to be improved. While the core of SIMD ISAs fulfill the same purpose, and are often interchangeable in functionality, none of the ISAs are truly compatible as a whole. These small ISA differences could be abstracted away elegantly, through a vendor independent standardized SIMD extension to programming languages. A standard SIMD language extension would allow programmers to target multiple SIMD ISAs with a single SIMD parallelization method.

From a video codec perspective the main implementation complexity arises from the support for multiple bitdepths. For each function the possible block sizes and the input and intermediate precisions must be carefully considered to avoid overflows. This could be avoided by reducing the supported bitdepths, e.g., only support 8-bit and 12-bit instead of all the possible bit depths in between. Another possible solution is to specify the codec in standardized floating point operations. General purpose architectures have lately increased their floating point performance at a much faster rate than integer performance and in the latest architectures have even a higher floating point throughput. As the use of floating point numbers might also improve compression performance it is an interesting and promising direction for future work.

REFERENCES

[1] R. Lee, "Realtime MPEG Video via Software Decompression on a PA-RISC processor," in *Compcon '95.'Technologies for the Information Superhighway'*, pp. 186–192, 1995.

[2] K. Mayer-Patel, B. C. Smith, and L. A. Rowe, "The Berkeley Software MPEG-1 Video Decoder," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 1, pp. 110–125, feb 2005.

[3] V. Lappalainen, T. Hamalainen, and P. Liuha, "Overview of Research Efforts on Media ISA Extensions and their Usage in Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 660–670, Aug. 2002.

[4] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1649–1668, Dec. 2012.

[5] J. Ohm, G. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the Coding Efficiency of Video Coding Standards–Including High Efficiency Video Coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669–1684, 2012.

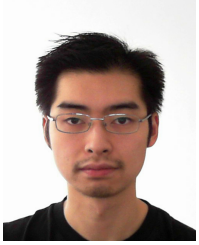
[6] M. Taylor, "A Landscape of the New Dark Silicon Design Regime," *IEEE Micro*, vol. 33, pp. 8–19, Sept 2013.

[7] M. Flynn, "Very High-speed Computing Systems," *Proceedings of the IEEE*, vol. 54, pp. 1901–1909, Dec. 1966.

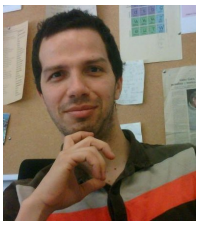
[8] R. B. Lee, "Accelerating Multimedia with Enhanced Microprocessors," *IEEE Computer*, vol. 15, pp. 22–32, April 1995.

[9] V. Bhaskaran, K. Konstantinides, R. B. Lee, and J. P. Beck, "Algorithmic and Architectural Enhancements for Real-time MPEG-1 Decoding on a General Purpose RISC Workstation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, pp. 380–386, Oct. 1995.

[10] A. Peleg and U. Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, vol. 16, pp. 42–50, Aug. 1996.
[11] K. Diefendorff, P. Dubey, R. Hochsprung, and H. Scales, "AltiVec Extension to PowerPC Accelerates Media Processing," *IEEE Micro*, vol. 20, pp. 85–95, April 2000.
[12] S. Thakkar and T. Huff, "The Internet Streaming SIMD extensions," *IEEE Computer*, vol. 32, pp. 26–34, Dec. 1999.
[13] S. K. Raman, V. Pentkovski, and J. Keshav, "Implementing Streaming SIMD Extensions on the Pentium III Processor," *IEEE Micro*, vol. 20, pp. 47–57, Aug. 2000.
[14] Intel, "Intel Advanced Vector Extensions Programming Reference," Tech. Rep. 319433-011, Intel, June 2011.
[15] Intel, "Intel Architecture Instruction Set Extensions Programming Reference," Tech. Rep. 319433-015, Intel, July 2013.
[16] X. Zhou, E. Q. Li, and Y.-K. Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," in *Proc. of SPIE Conf. on Image and Video Communications and Processing*, 2003.
[17] Y.-K. Chen, E. Q. Li, X. Zhou, , and S. Ge, "Implementation of H.264 Encoder and Decoder on Personal Computers," *Journal of Visual Communications and Image Representations*, 2006.
[18] V. Iverson, J. McVeigh, and B. Reese, "Real-time H.264-AVC Codec on Intel Architectures," in *International Conference on Image Processing, ICIP '04*, vol. 2, pp. 757–760 Vol.2, 24-27 2004.
[19] L. Yan, Y. Duan, J. Sun, and Z. Guo, "Implementation of HEVC Decoder on x86 Processors with SIMD Optimization," in *IEEE Visual Communications and Image Processing (VCIP)*, pp. 1–6, 2012.
[20] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC Complexity and Implementation Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1685–1696, 2012.
[21] F. Bossen, "On software complexity: decoding 1080p content on a smartphone," Tech. Rep. JCTVC-K0327, October 2012.
[22] B. Bross, M. Alvarez-Mesa, V. George, C. C. Chi, T. Mayer, B. Juurlink, and T. Schierl, "HEVC Real-time Decoding," in *Proceedings of SPIE 8856, Applications of Digital Image Processing XXXVI*, August 2013.
[23] "Write Combining Memory Implementation Guidelines," Tech. Rep. 244422-001, Intel, 1998.
[24] M. Budagavi, A. Fuldseth, G. Bjontegaard, V. Sze, and M. Sadafale, "Core Transform Design in the High Efficiency Video Coding (HEVC) Standard," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, pp. 1029–1041, Dec. 2013.
[25] W.-H. Chen, C. Smith, and S. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform," *IEEE Transactions on Communications*, vol. 25, no. 9, pp. 1004–1009, 1977.
[26] A. Norkin, G. Bjontegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, and G. Van der Auwera, "HEVC Deblocking Filter," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1746–1754, Dec 2012.
[27] C.-M. Fu, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park, and W.-J. Han, "Sample Adaptive Offset in the HEVC Standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1755–1764, Dec. 2012.
[28] G. Sullivan and S. Estrop, "Recommended 8-Bit YUV Formats for Video Rendering," 2002, update 2008.
[29] F. Bossen, "Common test conditions and software reference configurations," Tech. Rep. L1100, JCTVC, January 2013.
[30] H. Hoffman, A. Kouadio, Y. Thomas, and M. Visca, "The Turin Shoots," in *EBU Tech-i*, no. 13, pp. 8–9, European Broadcasting Union (EBU), September 2012.
[31] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1827–1838, 2012.



Chi Ching Chi received the BSc degree in electrical engineering in 2008 and MSc degree in computer engineering in 2010 from Delft University of Technology, the Netherlands. Since 2010, he is working towards the PhD degree in the Embedded Systems Architecture group at TU Berlin, Germany. His research interest include multi- and many-core architectures, operating systems, parallel programming models and languages, and video compression applications.



Mauricio Alvarez-Mesa received the MSc degree in Electronic Engineering in 2000 from University of Antioquia, Medellin, Colombia and the PhD degree in Computer Science in 2011 from Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. From 2006 to 2011 he was an adjunct lecturer at UPC. He was a summer intern at IBM Haifa Research labs, Israel in 2007, and a research visitor at Technische Universität Berlin (TU Berlin), Berlin, Germany in 2011. In January 2012 he joined the Multimedia Communications group at the Fraunhofer Institut HHI in Berlin and the Embedded Systems Architecture group at TU Berlin. His research interest includes parallel computing, computer architecture and video coding.

Benjamin Bross (S'11) received the Dipl.-Ing. degree in electrical engineering from RWTH University Aachen, Germany in 2008. During his studies he was working on three-dimensional image registration in medical imaging and on decoder side motion vector derivation in H.264/AVC. He is currently with the image processing group at Fraunhofer Institute for Telecommunications – Heinrich Hertz Institute, Berlin, Germany. His research interests include motion estimation/prediction, residual coding and contributions to the evolving High Efficiency Video Coding standard, H.265/HEVC. Since 2010, he is coordinating core experiments for the development of H.265/HEVC and co-chairing the editing ad hoc group. In July 2011, he was appointed as the Editor of the H.265/HEVC video coding standard.

Ben Juurlink is professor of Embedded Systems Architectures of the Electrical Engineering and Computer Science faculty of TU Berlin. He has an MSc degree from Utrecht University (NL) and a PhD degree from Leiden University (NL). In 1997-1998 he worked as a post-doctoral research fellow at the Heinz Nixdorf Institute in Paderborn (DE). From 1998 to 2009 he was a faculty member in the Computer Engineering laboratory of Delft University of Technology (NL). His research interests include multi- and manycore processors, instruction-level parallel and media processors, low-power techniques, and hierarchical memory systems. He has (co-)authored more than 100 papers in international conferences and journals and received a best paper award at the IASTED PDCS conference in 2002. He has been the leader of several national projects, work package leader in several European projects, and is currently coordinator of the EU FP7 project LPGPU (lpgpu.org). He is a senior member of the IEEE, a member of the ACM, and a member of the HiPEAC NoE. He served in many program committees, is area editor of the journal Microprocessors and Microsystems: Embedded Hardware Design (MICPRO), and is general co-chair of the HiPEAC 2013 conference.

Thomas Schierl received the Diplom-Ingenieur degree in Computer Engineering from the Berlin University of Technology (TUB), Germany in December 2003 and the Doktor der Ingenieurwissenschaften (Dr.-Ing.) degree in Electrical Engineering and Computer Science from Berlin University of Technology (TUB) in October 2010. He has been with Fraunhofer Institute for Telecommunications — HHI since end of 2004. Since 2010, Thomas is head of the Multimedia Communications Group in the Image Processing Department of Fraunhofer HHI, Berlin. Thomas is the co-author of various IETF RFCs, beside others he is author of the IETF RTP Payload Format for H.264 SVC (Scalable Video Coding) as well as for HEVC. In the ISO/IEC MPEG group, Thomas is co-editor of the MPEG Standard on Transport of H.264 SVC, H.264 MVC and HEVC over MPEG-2 Transport Stream. Thomas is also a co-editor of the AVC File Format. In 2007, he visited the Image, Video, and Multimedia Systems group of Prof. Bernd Girod at Stanford University, CA, USA for different research activities. Thomas' research interests currently focus on mobile media streaming and content delivery.