SIMD Architectural Enhancements to Improve the Performance of the 2D Discrete Wavelet Transform

Asadollah Shahbahrami[1, 2]
[1]Computer Engineering Laboratory
Delft University of Technology
2628 CD Delft, The Netherlands
A.Shahbahrami,B.H.H.Juurlink@TUDelft.nl

Ben Juurlink[1]
[2]Department of Computer Engineering
Faculty of Engineering
University of Guilan
Rasht, Iran

***Abstract*—The 2D Discrete Wavelet Transform (DWT) is a time-consuming kernel in many multimedia applications such as JPEG2000 and MPEG-4. The 2D DWT consists of horizontal filtering along the rows followed by vertical filtering along the columns. The vertical filtering is easy to vectorize (assuming row-major order), but to vectorize the horizontal filtering many overhead instructions are required. In this paper we propose some SIMD architectural enhancements, such as the MAC operation, extended subwords, and the matrix register file technique, to develop high-performance implementations of the 2D DWT on SIMD architectures. The MAC operation performs four 32-bit single-precision floating-point multiplications with accumulation. The matrix register file allows to load data stored consecutively in memory to a column of the register file, where a column corresponds to corresponding subwords of different registers. These techniques avoid the need of data rearrangement instructions. In addition, in order to avoid data type conversion instructions, the extended subword technique is applied for the $(5,3)$ lifting transform. Extended subwords use registers that are wider than the packed format used to store the data. These techniques provide speedups of up to 2.90 and 1.32 for the $(5,3)$ lifting and Daub-4 transforms, respectively.**

***Keywords*-Parallelization, SIMD Architectures, DWT.**

I. Introduction

The Discrete Wavelet Transform (DWT) is an important function in many digital signal processing applications such as JPEG2000 and MPEG-4. This is because it provides a multi-resolution representation of a signal and achieves higher compression ratios for multimedia data compression standards than other transforms such as the Discrete Cosine Transform (DCT) [1]. However, the DWT is much more computationally intensive than other functions. Our results show that the DWT consumes on average 46% of the total JPEG2000 encoding time for lossless compression and even 68% for lossy compression. Results presented by other researchers [2] also show that the DWT consumes a significant fraction of the total JPEG2000 encoding time.

In order to reduce complexity and improve performance, several researchers [3], [4] have proposed hardware implementations of the DWT. Programmable processors, however, are preferable to special-purpose hardware because they are more flexible, enable different transforms to be employed, and allow various filter bank lengths and various transform levels. Hence in this paper we focus on general-purpose, programmable SIMD processors.

In order to develop high-performance implementations of the 2D DWT on General-Purpose Processors (GPPs), several researchers [5], [6], [7] have vectorized it using the SIMD instructions supported by multimedia extensions such as SSE [8], [9]. Chaver et al. [5] used SSE and the Cohen, Daubechies and Feauveau 9/7 filter [10] (CDF-9/7). They focused on automatic vectorization. The Intel compiler, however, can only vectorize simple loops, and therefore some manual code modifications had to be performed. Furthermore, only horizontal filtering could be automatically vectorized (they assumed column-major order). In [6] they have vectorized vertical filtering of CDF-9/7 by hand using built-in SSE functions. In order to do so, however, an additional data transposition stage was required, which reduces the benefits of SIMD vectorization. Kutil [7] has implemented the $(9,7)$ lifting scheme using built-in SSE functions. He proposed a single loop approach to SIMD vectorization. In this approach horizontal and vertical filtering are combined into a single loop. The single-loop approach requires a buffer whose size is equal to 16 rows of data. If this buffer does not fit in the cache, the temporal locality will be reduced.

We have implemented both horizontal and vertical filtering of the Daubechies' transform with four coefficients [11], [12] (Daub-4) and the $(5,3)$ lifting scheme ($(5,3)$ *lifting*) [13] transform using SSE and MMX [14] extensions (we assume row-major order). Table I shows the number of dynamic instructions of the horizontal and vertical filtering and their ratio for the $(5,3)$ lifting and Daub-4 transforms for an $N \times M$ image. The number of executed instructions for the vertical filtering is 1.40 and 1.30 times smaller than the number of executed instructions for the horizontal filtering for the $(5,3)$ lifting scheme and Daub-4 transforms, respectively. This is because the horizontal filtering cannot be vectorized efficiently, while the vertical filtering can. To vectorize the horizontal filtering, overhead instructions are

This research was supported in part by the Netherlands Organization for Scientific Research (NWO).

Transforms	**Horizontal filtering**	**Vertical filtering**	**Ratio**
	# Dynamic ins.	# Dynamic ins.	Col. 2 / Col. 3
(5, 3) Lifting	$5 + (5 + \frac{31*M}{8}) * N$	$6 + (5 + \frac{22*M}{4}) * \frac{N}{2}$	1.40
Daub-4	$4 + (4 + \frac{48*M}{8}) * N$	$4 + (4 + \frac{37*M}{4}) * \frac{N}{2}$	1.30

Table I
THE NUMBER OF DYNAMIC INSTRUCTIONS OF THE SIMD IMPLEMENTATIONS OF THE HORIZONTAL AND VERTICAL FILTERING AND ALSO THEIR RATIO FOR DIFFERENT TRANSFORMS FOR AN $N \times M$ IMAGE.

needed. These instructions represent the overhead necessary to put data in a format suitable to SIMD operations, such as *packing/unpacking* and data *re-shuffling* instructions. In addition, in the MMX implementation of the $(5, 3)$ lifting, there is a mismatch between the storage and the computational formats. For instance, about 12.7% of the dynamic instructions are needed to convert the pixels from 8-bit to 16-bit values (data type conversion instructions).

Therefore, architectural enhancements, such as the *Multiply-ACcumulate* (MAC) operation and the *Matrix Register File* (MRF) technique to aid the efficient SIMD vectorization of horizontal filtering are proposed. A packed MAC instruction is used for the Daub-4 transform. The MAC operation performs four 32-bit single-precision floating-point multiplications with accumulation. The SSE/SSE2/SSE3 ISAs do not provide the MAC operation for floating-point numbers. Only a packed multiply and add instruction for fixed-point numbers is supported. The MRF allows to load data stored consecutively in memory to a column of the register file, where a column corresponds to corresponding subwords of different registers. This technique avoids the need of data rearrangement instructions. In addition, in order to avoid data type conversion instructions, the extended subword technique is applied for the $(5, 3)$ lifting transform. Extended subwords use registers that are wider than the packed format used to store the data.

We make the following contributions compared to other works.

- Although the MRF has been applied to fixed-point numbers using the MMX registers in [15], we apply this technique to floating-point numbers using the SSE register file.
- We propose the MAC operation and the MRF technique to avoid overhead instructions in the SIMD implementation of the horizontal filtering of the Daub-4 transform.
- In order to alleviate both overhead and data type conversion instructions in the horizontal filtering of the $(5, 3)$ lifting scheme, we propose the use of both the MRF and extended subwords.
- We propose the use of extended subwords to alleviate data type conversion instructions in the vertical filtering of the $(5, 3)$ lifting scheme.

This paper is organized as follows. Section II describes the discrete wavelet transform and the Modified MMX (MMMX) architecture that features the extended subwords and the Matrix Register File (MRF) techniques. Section III discusses the SIMD implementations of the 2D DWT using multimedia extensions and illustrates their limitations. The implementation of the proposed techniques is explained in Section IV followed by performance evaluation in Section V. Finally, conclusions are given in Section VI.

II. BACHGROUND

In this section we describe the discrete wavelet transform and how the MMX architecture has been enhanced with extended subwords and the MRF.

A. Discrete Wavelet Transform

A 2D DWT consists of *horizontal filtering* along the rows followed by *vertical filtering* along the columns. There are different approaches to implement both horizontal and vertical filtering such as traditional convolution-based and lifting scheme methods. The convolutional methods apply filtering by multiplying the filter coefficients with the input samples and accumulating the results. The Daub-4 transform is an example of this category. The lifting scheme has been proposed for the efficient implementation of the 2D DWT. This approach has three phases, namely: split, predict, and update. One example of this group is the integer-to-integer $(5, 3)$ lifting scheme [13]. In this paper, we have implemented both the Daub-4 and the $(5, 3)$ lifting transforms as an example of each category. The $(5, 3)$ lifting scheme is included because it has low computational complexity and performs reasonably well for lossy as well as lossless compression compared to other filters [13]. We remark that the $(5, 3)$ transform is included in Part 1 of the JPEG2000 standard. Furthermore, these transforms have been considered in many recent papers (e.g., [2], [6]). Although we have used these two transforms, the proposed techniques are general and equally applicable to other transforms.

B. MMMX Architecture

The MMMX architecture is MMX enhanced with extended subwords, the MRF, and a few general-purpose SIMD instructions that are not present in MMX. The employed techniques in the MMMX architecture are discussed briefly in the following section. More detail about this architecture can be found in [16], [11].

1) Extended Subwords: Image and video data is typically stored as packed 8-bit elements, but intermediate results usually require more than 8-bit precision. As a consequence, most 8-bit SIMD ALU instructions are wasted. In the SIMD extensions, the choice is either to be imprecise by using saturation operations at every stage, or to loose parallelism

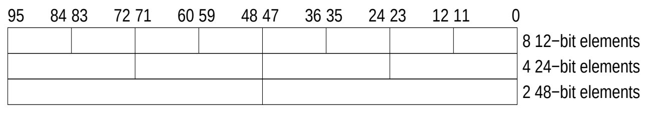


Figure 1. Different subwords in the media register file of the MMMX architecture.

by unpacking to a larger format. Using saturation instructions produces inaccurate results. This is because saturation is usually used at the end of computation. It is more precise to saturate once at the end of the computation rather than at every step of the algorithm. For instance, adding three signed 8-bit values $120 + 48 - 10$, using signed saturation at every step produces 117 and using signed saturation at the last step produces 127.

SIMD architectures support different packing, unpacking, and extending instructions to convert the different data types to each other. For example, the MMX/SSE architectures provide `packss`{`wb,dw,wb`} and `punpck` {`hbw,hwd,hdq,lbw,lwd,ldq`} instructions for data type conversions.

To avoid the data type conversion overhead and to increase parallelism, extended subwords are employed. This means that the registers are wider than the data loaded into them. Specifically, for every byte of data, there are four extra bits. This implies that MMMX registers are 96 bits wide, while MMX has 64-bit registers. Based on that, the MMMX registers can hold 2×48-bit, 4×24-bit, or 8×12-bit elements as is depicted in Figure 1.

2) The Matrix Register File: The ability to efficiently rearrange subwords within and between registers is crucial to performance. To overcome this problem, a matrix register file is employed, which allows data loaded from memory to be written to a column of the register file as well as to a row register. In the MMMX architecture, the MRF provides parallel access to 12-, 24-, and 48-bit subwords of the row registers that are horizontally located. This is similar to conventional SIMD architectures, which provide parallel access to 8-, 16-, and 32-bit data elements of media registers. In addition, the MRF provides parallel access to 12-bit subwords of the column registers that are vertically arranged.

Figure 2 illustrates an MRF with 12-bit subwords. It has eight row registers $3mxi, 0 \leq i \leq 7$ (corresponding to conventional media registers) and eight column registers $3mxci, 0 \leq i \leq 7$ (corresponding subwords in different row registers). Both row and column registers are 96 bits wide. Data loaded from memory can be written to a row register as well as to a column register. Seven 2:1 12-bit multiplexers are needed per register/row to select between row-wise and column-wise access. For example, for register `3mx0` it needs to be able to select between the most significant subword of the data for column-wise access and another subword in case of row-wise access. Multiplexers are not needed for the

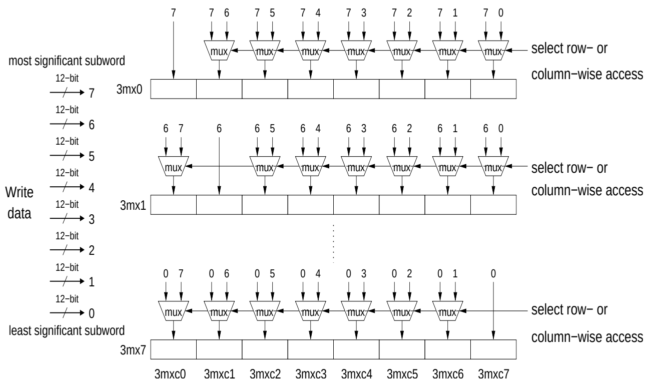


Figure 2. A matrix register file with 12-bit subwords. For simplicity, write and clock signals have been omitted.

subwords on the main diagonal.

Although the idea of the MRF has been applied to fixed-point numbers using the MMX registers in [15], we apply this technique to floating-point numbers using the SSE register file. In other words, the MMMX architecture is an SIMD fixed-point extension, while one goal of this work is to apply the MRF to the SIMD floating-point extension.

III. SIMD Implementation of the 2D DWT

In this section we discuss the vectorization of the Daub-4 transform as an example of convolution-based approaches and $(5, 3)$ lifting scheme as an example of lifting scheme methods.

A. Vectorization of the Daub-4 Transform

The Daub-4 processes single-precision floating-point values and applies filtering by multiplying the filter coefficients with the input samples and accumulating the results. Under a row-major image layout, it is relatively straightforward to vectorize vertical filtering using SSE instructions. This is because the elements that can be processed simultaneously are stored consecutively in memory. Consider, for example, the Daub-4 transform and let $x_{i,j}$ be the input samples and let $c_0, \ldots, c_3$ denote the lowpass filter coefficients. Figure 3 illustrates the data flow graph of the vertical filtering. As this figure shows four different input samples of each row are multiplied with one filter coefficient simultaneously. Each filter coefficient should be replicated in each of the four different subwords of a media register. After four multiplications of four consecutive rows with different coefficients, the results of each column are added to each other. Finally, four wavelet coefficients are calculated simultaneously. There are SIMD instructions in conventional SIMD extensions such as SSE for those operations.

Horizontal filtering is more difficult to vectorize, however. Figure 4 depicts the data flow graph of the horizontal filtering. As this figure shows four different input samples are multiplied with four different coefficients. The intermediate results are accumulated into one destination operand. In

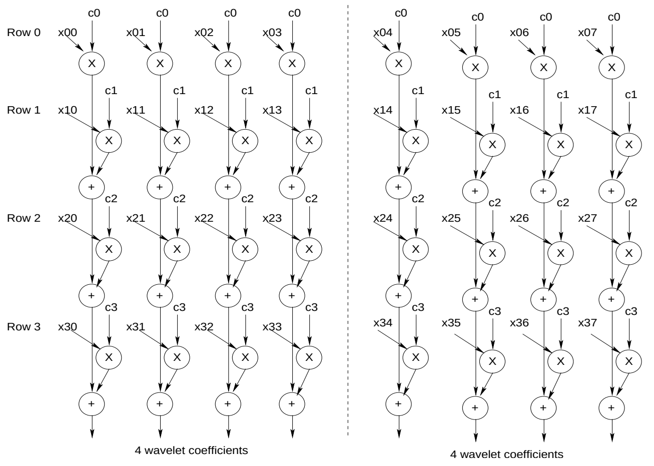


Figure 3. Data flow graph of the vertical filtering of the Daub-4 transform.

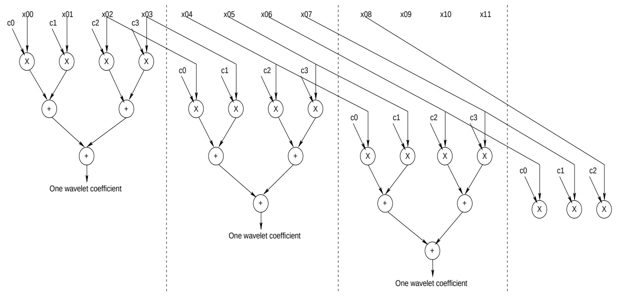


Figure 4. Data flow graph of the horizontal filtering of the Daub-4 transform.

other words, to map this figure to SIMD instructions, an MAC instruction would have been useful, but since SSE does not provide such an instruction, the elements should be rearranged so that, for example, the input samples $x_{i,2j}$, $x_{i,2j+2}$, $x_{2i,2j+4}$, and $x_{2i,2j+6}$ are stored consecutively in an SSE register. This requires many overhead (unpack) instructions. In other words, to vectorize the horizontal filtering, the matrix needs to be transposed twice. Transposition takes a significant amount of time, however. For example, 20 SIMD instructions are needed to transpose a 4×4 block of single-precision floating-point values using SSE instructions.

B. *Vectorization of the $(5, 3)$ Lifting Transform*

The lifting operation consists of three stages: split, predict, and update. First, the original 1D input signal is split into a subsequence consisting of the even-numbered input values $\{s_i^0\}$ and a subsequence containing the odd-numbered input values $\{d_i^0\}$. Thereafter, the prediction stage produces the highpass output values $\{d_i^1\}$ and the update stage generates the lowpass output values $\{s_i^1\}$.

The equations that are used in the prediction and update stage of the $(5, 3)$ lifting transform are given by:

$$d_i^1 = d_i^0 - \lfloor \frac{s_i^0 + s_{i+1}^0}{2} \rfloor \quad (1)$$

$$s_i^1 = s_i^0 + \lfloor \frac{d_{i-1}^1 + d_i^1 + 2}{4} \rfloor \quad (2)$$

The vectorization of the $(5, 3)$ lifting scheme is significantly different from the vectorization of Daub-4 transform for the following reasons. First, the $(5, 3)$ lifting transform uses integer arithmetic and hence its SIMD implementation employs MMX instructions. Second, in the MMX implementation there are no multiplication operations, since the input values need to be divided by powers of 2 which can be accomplished using shift operations. Third, because of its structure, the $(5, 3)$ lifting scheme is vectorized in a completely different way than the convolutional transforms.

In order to vectorize horizontal filtering, the data needs to be rearranged so that the even and odd subsequences are placed in different registers. Furthermore, because s_i^0 and s_{i+1}^0 have to be added, two copies of the even subsequence are required, one that starts with s_0^0 and one that starts with s_1^0. As was the case for the convolutional transforms, vertical filtering is easier to vectorize. In this case, the even and odd subsequences do not have to be split because they correspond to different rows. In addition, in the MMX implementation of the $(5, 3)$ lifting, there is a mismatch between the storage and the computational formats. About 12.7% of the number of dynamic instructions are needed to convert the pixels from 8-bit to 16-bit values (data type conversion instructions).

IV. Architectural Enhancements for SIMD Vectorization of the 2D DWT

In this section the SIMD architectural enhancements to enhance the performance of the 2D DWT are discussed. First, the SIMD implementations of the horizontal filtering of the Daub-4 transform using the MAC operation and the MRF are discussed. Second, the SIMD implementation of the $(5, 3)$ lifting transform using extended subwords and the MRF is explained.

A. *MAC Operation and the MRF Technique for the Daub-4 Transform*

The SSE ISA includes a packed multiply and add (`pmaddwd`) instruction for integers, but does not provide such an instruction for floating-point numbers. Providing an MAC unit that can perform a 32-bit single-precision floating-point multiplication with accumulation is a good solution to vectorize horizontal filtering of the convolution-based transform. As Figure 5 shows, multiplication of coefficients and input samples is possible without using overhead instructions and replication of coefficients. The `pmaddsd` (parallel multiply and add single-precision values to double-precision) performs an SIMD multiply of the four single-precision floating-point values in the source operand by the

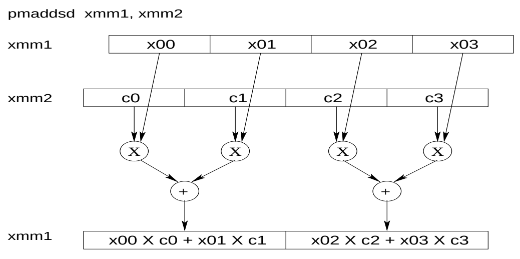


Figure 5. The structure of the `pmaddsd` instruction.

four single-precision floating-point values in the destination operand. The two high-order words are accumulated and stored in the upper doubleword of the destination operand, and the two low-order words are accumulated and stored in the lower doubleword of the destination operand.

The MRF is extended to floating-point numbers using the SSE register file. The SSE register file has eight 128-bit floating-point registers $xmm0, ..., xmm7$. Each register contains four single-precision floating-point values. The SSE MRF has eight row registers, the same as the normal register file, and four column registers $xcmm0, ..., xcmm3$. Figure 6 depicts the architecture of the modified register file with 32-bit subwords. This modified register file has two read ports and one write port that has been connected to a 128-bit SIMD floating-point unit. As this figure shows four registers ($xmm0$ to $xmm3$) can be accessed in both horizontal and vertical directions. Data loaded from memory can be written to a row register as well as to a column register. Three 2:1 32-bit multiplexers are needed in each row to select between row-wise and column-wise accesses. Only load-column instructions can write to a column register. Therefore, a transposition of a block stored in the memory can be performed using column-wise load instructions followed by normal store instructions. Four load-column instructions and four normal store instructions are needed to transpose a 4×4 block of single-point floating-point values using the modified register file, while 20 SSE instructions are required as was discussed in Section III. Each load-column instruction can load 128 bits from memory to a column register the same as the normal load and store instructions that are supported by the SSE extensions. For instance, the `movaps` instruction transfers 128 bits of packed data from memory to a row register. The modified register file is used for vectorization of the horizontal filtering of the Daub-4 transform.

B. Extended Subwords and the MRF for the $(5, 3)$ Lifting Transform

In order to evaluate if extended subwords can be used to improve the performance of the $(5, 3)$ lifting transform, the minimum and maximum wavelet coefficient and intermediate result for a 5-level decomposition have been determined. As input, the well-known “Lena” image as well as randomly generated images with 7 to 10 bits per pixel (bpp) have been employed. The results show that a 12-bit data format is sufficient for a 5-level decomposition of images of up to 10 bpp. This means that the extended subwords technique can be employed in order to exploit more Data-Level Parallelism (DLP) in the $(5, 3)$ lifting transform.

Therefore, extended subwords and the MRF are used to vectorize the horizontal filtering of the $(5, 3)$ lifting transform, while the extended subword is employed for the implementation of vertical filtering. Figure 7 illustrates how the MRF technique is used to reorder the input data. Eight load-column instructions are used to load the input sequence into the $3mxc0, 3mxc1, ..., 3mxc7$ column registers of the MMMX architecture. To provide correct arrangement of even and odd values according to Equation (1) and Equation (2), an offset, which is a multiple of 6 bytes for each load-column instruction, is used. After eight load-column instructions, each row register contains either even ($\{s_i^0\}$) or odd ($\{d_i^0\}$) values. Thereafter, the SIMD ALU instructions can be used to process the row registers. In this way, the extended subwords technique provides 8-way parallelism in both horizontal and vertical filtering.

V. Performance Evaluation

In this section we evaluate the proposed techniques by comparing the performance of the SIMD implementations that employ the SIMD architectural enhancements to the performance of the MMX and SSE implementations on a single issue processor.

A. Evaluation Environment

In order to evaluate the SIMD architectural enhancements, we have used the `sim-outorder` simulator of the SimpleScalar toolset [17]. We have synthesized MMX/SSE and MMMX instructions using the 16-bit annotate field, which is available in the instruction format of the PISA ISA. More detail about our extension to the SimpleScalar toolset can be found in [18].

The main objective is to compare the performance of the MMX and SSE extensions without the proposed techniques to the those extensions with the SIMD architectural enhancements. The main parameters of the modeled processors are depicted in Table II. The latency and throughput of SIMD instructions are set equal to the latency and throughput of the corresponding scalar instructions. This is a conservative assumption given that the SIMD instructions perform the same operation but on narrower data types. The latency and throughput of SIMD multiplier units are set to 3 and 1 respectively, the same as in the Pentium 3 processor. The latency of SIMD multiplier units in the Pentium 4 processor is 8 cycles.

For both horizontal and vertical filtering of the $(5, 3)$ lifting transform, two SIMD implementations using MMX

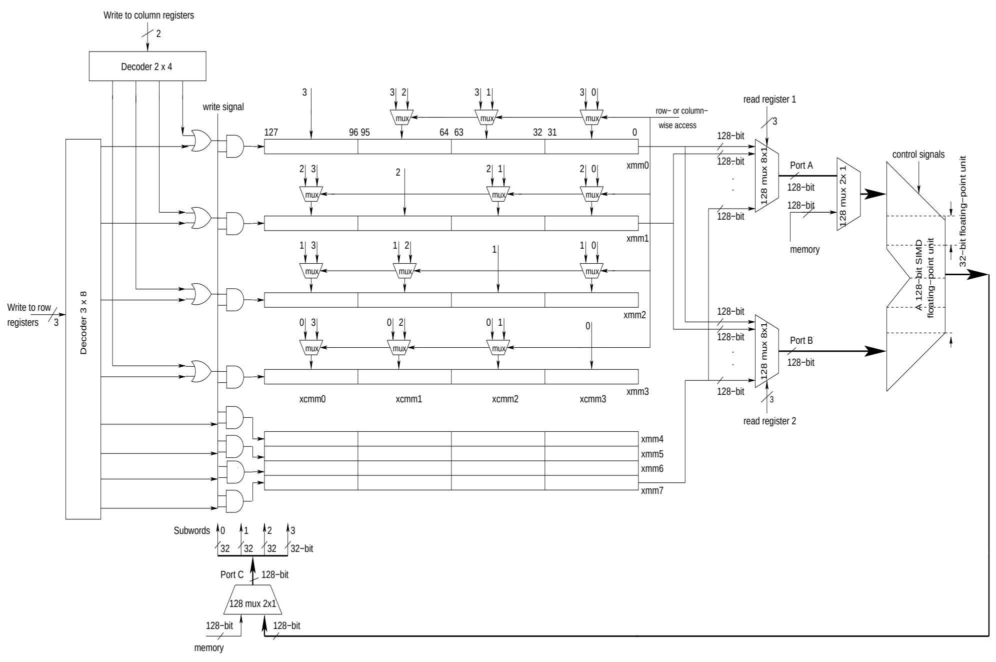
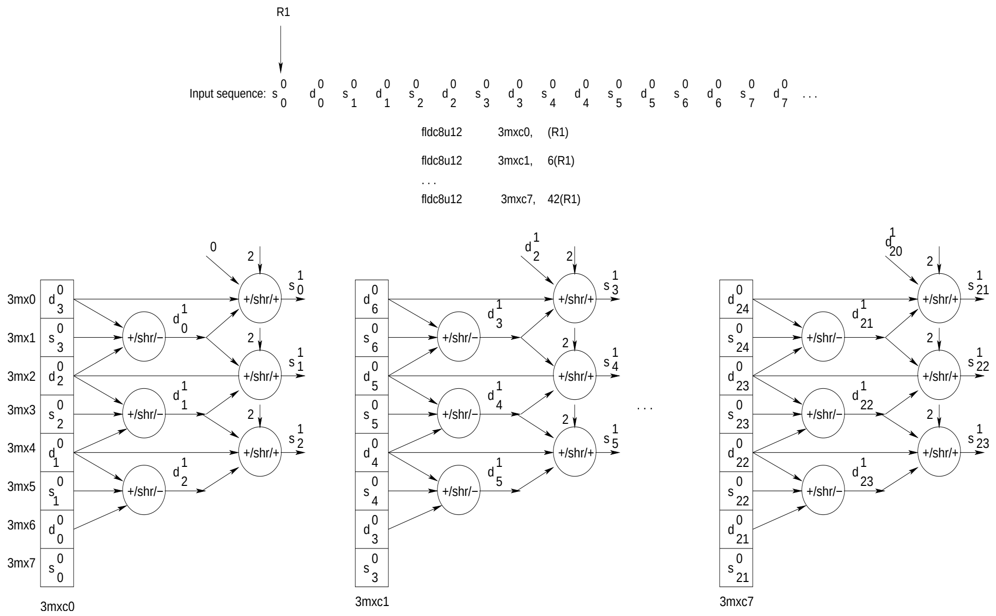


Figure 6. A matrix register file with eight 128-bit registers, two read ports, and one write port. Four registers can be accessed in row-wise as well as column-wise. The modified register file is connected to a 128-bit SIMD floating-point unit.



R1
Input sequence: s^0_0 d^0_0 s^0_1 d^0_1 s^0_2 d^0_2 s^0_3 d^0_3 s^0_4 d^0_4 s^0_5 d^0_5 s^0_6 d^0_6 s^0_7 d^0_7 ...
fldc8u12 3mxc0, (R1)
fldc8u12 3mxc1, 6(R1)
...
fldc8u12 3mxc7, 42(R1)
3mx0
3mx1
3mx2
3mx3
3mx4
3mx5
3mx6
3mx7
+/shr/−
+/shr/+
3mxc0
3mxc1
3mxc7

Figure 7. Vectorization of the horizontal filtering of the $(5, 3)$ lifting scheme using the matrix register file and extended subwords techniques.

Parameter	Value
Issue width	1
Integer ALU, SIMD ALU	1
Integer MULT, SIMD MULT	1
L1 Instruction cache	512 set, direct-mapped 64-byte line LRU, 1-cycle hit, total of 32 KB
L1 Data cache	128 set, 4-way, 64-byte line, 1-cycle hit, total of 32 KB
L2 Unified cache	1024 set, 4-way, 64-byte line, 6-cycle hit, total of 256 KB
Main memory latency	18 cycles for first chunk, 2 thereafter
Memory bus width	16 bytes
RUU (register update unit) entries	64
Load-store queue size	8
Execution	out-of-order

Table II
PROCESSOR CONFIGURATION.

and MMMX instructions have been implemented and simulated. In the MMMX implementation of horizontal filtering, both extended subwords and the MRF have been used, while in the vertical filtering only extended subwords have been used. In addition, three SIMD implementations, namely SSE, SSE-MAC, and SSE-MRF of the horizontal filtering of the Daub-4 transform have been implemented and simulated. The SSE version uses overhead instructions to reorder the data. In the SSE-MAC implementation, the proposed MAC operation has been used, while in the SSE-MRF implementation the modified SSE register file has been employed. The performance obtained by the MMMX and SSE-MAC as well as SSE-MRF implementations is compared to the performance attained by the MMX and SSE implementations, respectively. In other words, the SIMD implementations of the MMX and SSE are used as reference implementations.

B. Performance Evaluation Results

Figure 8 depicts the speedups of the MMMX implementation of the horizontal and vertical filtering of the $(5, 3)$ lifting, SSE-MAC and SSE-MRF implementations of the horizontal filtering of the Daub-4 transform over MMX and SSE, respectively, as well as the ratio of committed instructions for an image size of 480×480 on a single issue processor. The MMMX implementation of the horizontal filtering of the $(5, 3)$ lifting transform is 2.90 times faster than the MMX implementation, while the MMMX implementation of the vertical filtering is 2.03 times faster than the MMX implementation. The reason why the speedup of the horizontal filtering is larger than the vertical filtering is that in the MMMX implementation of the horizontal filtering both techniques, extended subwords and the MRF, have been used, but in the MMMX implementation of the vertical filtering only extended subwords have been used.

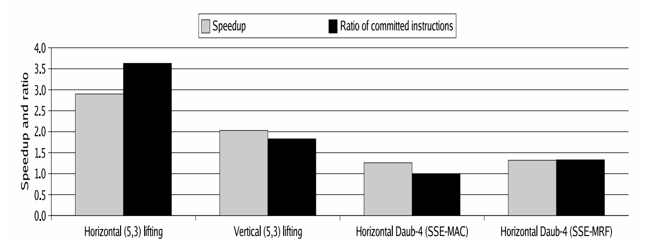


Figure 8. Speedups of the MMMX, SSE-MAC and SSE-MRF implementations over MMX and SSE, respectively, as well as the ratio of committed instructions.

The speedup of SSE-MAC and SSE-MRF is 1.26 and 1.32, respectively. The ratio of committed instructions is 1.00 and 1.33. This means that the MAC operation does not reduce the number of committed instructions. SSE-MAC executes eight SIMD and four scalar instructions in the inner loop to calculate two wavelet coefficients in each iteration. SSE executes 44 SIMD and four scalar instructions in the inner loop to calculate eight wavelet coefficients in each iteration. This means that in the SSE-MAC implementation, four SIMD and two scalar instructions are needed to calculate one wavelet coefficient, while in the SSE implementation, 5.5 SIMD and 0.5 scalar instructions are required to calculate one wavelet coefficient. As a result, SSE-MAC reduces the number of SIMD instructions, while it increases the number of scalar instructions. The latencies of the scalar instructions, which are used for incrementing or decrementing index and address values, however, are shorter than the latencies of the SIMD multiplication instructions. This is the reason why SSE-MAC yields a speedup of 1.26. In order to reduce the number of the scalar instructions in the SSE-MAC implementation, the inner loop has been unrolled four times. This unrolled version yields a speedup of 1.28 over SSE and reduces the number of committed instructions by 1.33x.

Table III shows the dynamic number of instructions for both horizontal and vertical filtering and their ratio for $(5, 3)$ lifting and Daub-4 transforms after using the proposed techniques for an $N \times M$ image. The ratio of the number of executed instructions of the horizontal filtering over the executed instructions of the vertical filtering is 0.85 and 0.97 for $(5, 3)$ lifting and Daub-4 transforms, respectively. Comparing Table I and Table III shows that after applying the SIMD architectural enhancements, the ratio of executed instructions is reduced from 1.40 and 1.30 to 0.85 and 0.97, respectively.

VI. CONCLUSIONS

In this paper we have focused on high-performance implementations of the 2D DWT on SIMD architectures. We have discussed the limitations of the SIMD implementations

Transforms	**Horizontal filtering**	**Vertical filtering**	**Ratio**
	# Dynamic ins.	# Dynamic ins.	Col. 2 / Col. 3
$(5,3)$ Lifting	$6+(4+\frac{51*M}{48})*N$	$5+(4+\frac{20*M}{8})*\frac{N}{2}$	0.85
Daub-4	$4+(4+\frac{36*M}{8})*N$	$4+(4+\frac{37*M}{4})*\frac{N}{2}$	0.97

Table III
NUMBER OF DYNAMIC INSTRUCTIONS OF THE SIMD IMPLEMENTATION OF BOTH HORIZONTAL AND VERTICAL FILTERING OF THE $(5,3)$ LIFTING AND DAUB-4 TRANSFORMS AFTER USING THE PROPOSED TECHNIQUES FOR AN $N \times M$ IMAGE.

of the lifting scheme and the convolution-based transforms using the MMX and SSE extensions. The vertical filtering is relatively straightforward to vectorize, while horizontal filtering requires to rearrange subwords within a register. Mainly because of this overhead, the speedups obtained from SIMD vectorization of the horizontal filtering are smaller than the speedups obtained for the vertical filtering. In addition, in the SIMD implementation of the lifting lifting, there is a mismatch between the storage and computational formats. In order to develop high-performance SIMD implementations, some architectural enhancements have been proposed in this work. A MAC unit has been proposed for SIMD implementation of the horizontal filtering of the Daub-4 transform. The matrix register file that already used for fixed-point numbers has been extended to floating-point numbers using the SSE register file. In addition, in order to avoid data type convertion instructions in the SIMD implementations of the lifting transforms, extended subwords have been used. The extended subwords and the MRF techniques have been used in the horizontal filtering of the $(5,3)$ lifting transform, while only the extended subwords technique has been employed in the vertical filtering. These techniques provide speedups of 2.90 and 2.03 for horizontal and vertical filtering, respectively. The modified SSE register file improves the performance of the horizontal filtering of the Daub-4 transform by a factor of 1.32, while the MAC operation yields a speedup of 1.26.

REFERENCES

[1] M. Rabbani and R. Joshi, "An Overview of the JPEG2000 Still Image Compression Standard," *Signal Processing: Image Communication*, vol. 17, no. 1, pp. 3–48, January 2002.

[2] S. Chatterjee and C. D. Brooks, "Cache Efficient Wavelet Lifting in JPEG 2000," in *Proc. IEEE Int. Conf. on Multimedia*, August 2002, pp. 797–800.

[3] P. P. Dang and P. M. Chau, "Reduce Complexity Hardware Implementation of Discrete Wavelet Transform for JPEG 2000 Standard," in *Proc. IEEE Int. Conf. on Multimedia and Expo*, August 2002, pp. 321–324.

[4] M. Ferretti and D. Rizzo, "A Parallel Architecture for the 2D Discrete Wavelet Transform with Integer Lifting Scheme," *Journal of VLSI Signal Processing*, vol. 28, pp. 165–185, 2001.

[5] D. Chaver, C. Tenllado, L. Pinuel, M. Prieto, and F. Tirado, "2-D Wavelet Transform Enhancement on General-Purpose Microprocessors: Memory Hierarchy and SIMD Parallelism Exploitation," in *Proc. Int. Conf. on the High Performance Computing*, December 2002.

[6] ——, "Vectorization of the 2D Wavelet Lifting Transform Using SIMD Extensions," in *Proc. 17th IEEE Int. Symp. on Parallel and Distributed Image Processing and Multimedia*, 2003.

[7] R. Kutil, "A Single-Loop Approach to SIMD Parallelization of 2D Wavelet Lifting," in *Proc. 14th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing*, 2006, pp. 413–420.

[8] S. K. Raman, V. Pentkovski, and J. Keshava, "Implementing Streaming SIMD Extensions on the Pentium 3 Processor," *IEEE Micro*, vol. 20, no. 4, pp. 47–57, July-August 2000.

[9] S. Thakkar and T. Huff, "The Internet Streaming SIMD Extensions," *Intel Technology Journal*, pp. 1–8, 1999.

[10] A. Cohen, I. Daubechies, and J. C. F. Eauveau, "Biorthogonal Bases of Compactly Supported Wavelets," *Communications on Pure and Appl. Math.*, vol. 45, no. 5, pp. 485–560, June 1992.

[11] A. Shahbahrami, "Avoiding Conversion and Rearrangement Overhead in SIMD Architectures," Ph.D. dissertation, Delft University of Technology, September 2008.

[12] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "Implementing the 2D Wavelet Transform on SIMD-Enhanced General-Purpose Processors," *IEEE Trans. on Multimedia*, vol. 10, no. 1, pp. 43–51, January 2008.

[13] D. M. Adams and F. Kossentini, "Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis," *IEEE Trans. on Image Processing*, vol. 9, no. 6, pp. 1010–1024, June 2000.

[14] A. Peleg, , and U. Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, vol. 16, no. 4, pp. 42–50, August 1996.

[15] A. Shahbahrami, B. Juurlink, and S. Vassiliadis, "Versatility of Extended Subwords and the Matrix Register File," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 5, no. 1, May 2008.

[16] ——, "Matrix Register File and Extended Subwords: Two Techniques for Embedded Media Processors," in *Proc. 2nd ACM Int. Conf. on Computing Frontiers*, May 2005.

[17] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, February 2002.

[18] B. Juurlink, D. Borodin, R. J. Meeuws, G. T. Aalbers, and H. Leisink, "The SimpleScalar Instruction Tool (SSIT) and the SimpleScalar Architecture Tool (SSAT)," Available via `http://ce.et.tudelft.nl/˜shahbahrami/`